UNIX

PORADNIK UŻYTKOWNIKA

Lech S. Borkowski

4 lipca 2003

Spis treści

Wstęp

Książka jest zbiorem dokumentacji i porad dla początkujących i średnio zaawansowanych użytkowników systemu operacyjnego Unix. Można ją wykorzystywać jako materiał wstępnego kursu Uniksa lub jako poradnik.

Notacja

Czcionka pogrubiona używana jest do oznaczenia nazw klawiszy i ich kombinacji w tekście. `Czcionką o stałej szerokości` pisane są nazwy poleceń, opcji, zawartość plików, wyniki poleceń. *`Czcionką pochyłą o stałej szerokości`* pisane są nazwy i teksty, które należy zastąpić własnymi wartościami. Znak `$` używany jest jako tekst zgłoszenia w powłoce Bourne'a. W nawiasach `[]` umieszczone są opcjonalne elementy poleceń.

Początki systemu

Od 1965 r. Bell Laboratories (część koncernu AT&T), MIT i General Electric wspólnie pracowały nad projektem Multics. Miał to być nowoczesny, wieloprocesorowy system operacyjny, obsługujący jednocześnie wielu użytkowników i działający w oparciu o hierarchiczny system plików. W 1969 r. AT&T wycofało się z projektu z powodu niezadowolających postępów prac. Prace kontynuowano w Bell Labs na komputerach serii PDP wyprodukowanych przez Digital Equipment Corporation. Pierwsza wersja Uniksa została uruchomiona na PDP-7. Jej autorami byli Ken Thompson, Dennis Ritchie, Joe Ossanna i Doug McIlroy. Nazwę Unix zaproponował Brian Kernighan. Był to żart językowy z Multicsa.

W 1971 r. system działał na PDP-11, mającym 16 kilobajtów pamięci operacyjnej i dysk o pojemności 512 kilobajtów. W 1972 r. w drugim wydaniu podręcznika programisty systemu Unix napisano, że zainstalowano już 10 kopii Uniksa i spodziewane są kolejne instalacje. W 1973 r. Ritchie i Thompson napisali od nowa jądro systemu w języku C, zrywając tym samym z tradycją pisania oprogramowania systemowego w asemblerze. Był to początek systemu w takiej postaci, w jakiej znamy go dzisiaj. Od 1974 r. Unix zaczął pojawiać się w uczelnianych centrach komputerowych, a później również w firmach. Unix był stosowany na coraz szerszą skalę również w AT&T.

O sukcesie Uniksa przesądziło kilka czynników. Fakt, że system został napisany w języku C, a nie w asemblerze, ułatwiał przystosowanie go do różnych platform sprzętowych. Unix bardzo ułatwia współdziałanie wielu różnych narzędzi. Poszczególne programy nie muszą być skomplikowane. Aby rozwiązać większy problem, zwykle wystarczy wiedzieć, jak połączyć prostsze części w jedną całość. To samo zadanie zwykle można wykonać na wiele różnych sposobów.

Struktura systemu Unix

Unix ma strukturę warstwową. Wewnętrzną warstwą jest sprzęt obsługujący system operacyjny. Jądro systemu (ang. *kernel*), komunikuje się bezpośrednio z warstwą sprzętową i obsługuje programy użytkownika. Nie mają one bezpośredniego kontaktu z warstwą sprzętową. Powinny jedynie wiedzieć, jak komunikować się z jądrem, a zadaniem jądra jest świadczenie określonych usług. Większość poprawnie napisanych programów użytkownika ma postać niezależną od szczegółów sprzętowych, dzięki czemu łatwo można je przenieść na inne platformy sprzętowe.

Programy użytkownika oddziałują z jądrem za pomocą wywołań systemowych. Do usług świadczonych przez jądro należą m.in. operacje dostępu do plików (otwieranie, zamykanie, odczyt, zapis), tworzenie dowiązań lub uruchamianie plików, zmiana właściciela pliku lub katalogu, zmiana bieżącego katalogu, tworzenie, wstrzymanie lub zamknięcie procesu, udostępnianie urządzeń i ustawianie ograniczeń dotyczących wykorzystywania zasobów systemowych.

Unix jest systemem wielodostępnym i wielowątkowym. W systemie może być zalogowanych wielu użytkowników jednocześnie, z których każdy może uruchomić wiele programów. Zadaniem jądra systemu jest zarządzanie procesami i użytkownikami, a co za tym idzie, kontrola dostępu do sprzętu (cpu, pamięci, dysku i innych urządzeń wejścia/wyjścia).

Programy

Program (polecenie) komunikuje się z jądrem systemu w celu pobrania informacji o środowisku, w jakim wykonywany jest program i wykonania funkcji wywoływanych w programie. Program może być wykonywalnym plikiem powłoki (skrypt powłokowy), wbudowanym poleceniem powłoki lub plikiem skompilowanym z kodu źródłowego.

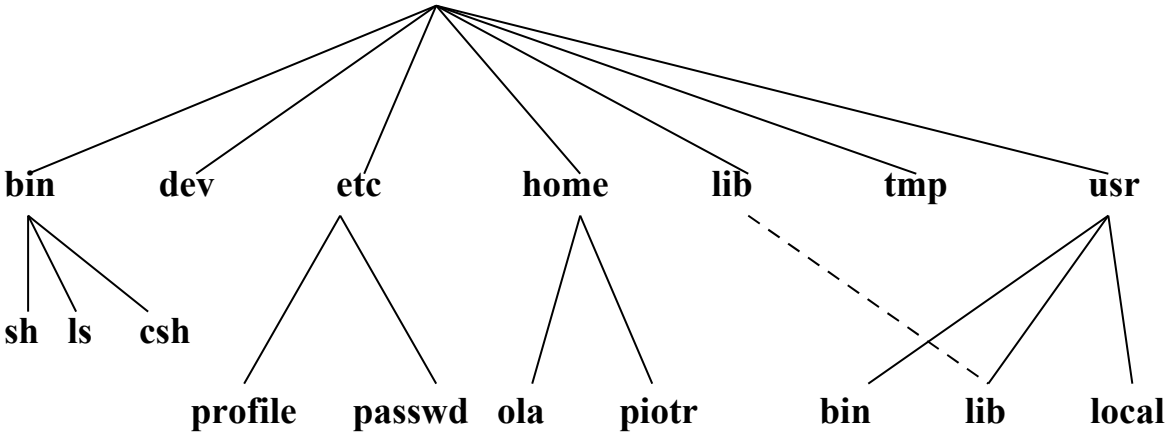
Powłoka jest interpreterem wiersza poleceń. Użytkownik komunikuje się z jądrem systemu za pośrednictwem powłoki. Skrypty powłokowe zapisywane są w formacie zwykłego tekstu i są wykonywane przez interpreter powłoki.

Programy systemowe są zwykle plikami binarnymi, powstałymi w wyniku kompilacji kodu źródłowego w języku C. Najczęściej są one ulokowane w następujących katalogach: `/bin`, `/sbin`, `/usr/bin`, `/usr/local/bin` i obsługują polecenia systemowe.

System plików

System plików Uniksa ma postać odwróconego drzewa. Na szczycie hierarchii znajduje się katalog główny systemu, oznaczony znakiem `/`.

Rysunek. Typowa struktura systemu plików Uniksa. Linia przerywana oznacza możliwość stworzenia dowiązania symbolicznego



Każdy węzeł drzewa jest plikiem lub katalogiem plików. Katalog może zawierać zarówno pliki, jak i katalogi. Pliki i katalogi są identyfikowane przez podanie ścieżki dostępu w postaci pełnej lub niepełnej. Pełna nazwa ścieżki zaczyna się od katalogu głównego, oznaczonego `/`, i zawiera poszczególne gałęzi systemu plików, oddzielone od siebie znakiem `/`. Ostatnim elementem jest plik, na przykład:

```
/home/ola/programy/a.c
```

Ścieżka niepełna określa ścieżkę względem innego katalogu, najczęściej bieżącego, który oznaczany jest kropką. Jeżeli bieżącym katalogiem jest `/home/ola`, a użytkownik chce wskazać ten sam plik, co w ostatnim przykładzie, powinien napisać `./programy/a.c`. Katalog nadrzędny jest oznaczany dwiema kropkami. Aby wskazać położenie tego samego pliku, znajdując się np. w katalogu `/home/piotr`, należy podać ścieżkę w następującej postaci:

```
../ola/programy/a.c
```

Oto główne gałęzi drzewa katalogów typowego Uniksa:

Ścieżka	**Zawartość**
`/`	Katalog główny systemu; zwykle na dysku lokalnym, ale może także funkcjonować na dysku sieciowym lub dysku ram; zawiera pliki niezbędne do uruchomienia systemu i przygotowania go do zamontowania innych systemów plików; zawiera wszystkie narzędzia potrzebne do pracy w trybie pojedynczego użytkownika (ang. *single user mode*); zawiera narzędzia do naprawy uszkodzonego systemu i przywracania plików z kopii zapasowych

`/usr`	Zawiera pliki wykonywalne poleceń w katalogu `/usr/bin`, biblioteki w katalogu `/usr/lib`, strony pomocy man `/usr/man` i inne pliki, które nie ulegają zmianie podczas normalnej pracy systemu; dane tej części systemu nie muszą być przesyłane w sieci lokalnej, a administrator takiej sieci wprowadza zmiany tylko w kopii-matce tej gałęzi; uprawnienia w tej części systemu są często ograniczone jedynie do odczytu
`/var`	Zmienne zasoby systemu: zwykle poczta, usenet news, kolejki drukowania, pliki dzienników systemowych, sformatowane strony man i pliki tymczasowe
`/home`	Zasoby (konta) użytkowników

Poszczególne gałęzi systemu plików mogą być ulokowane w oddzielnych partycjach.

Katalogi, pliki i i-węzły

Nazwa każdego katalogu i pliku jest zapisana w katalogu nadrzędnym. Wyjątkiem jest katalog główny systemu, który jest katalogiem nadrzędnym dla samego siebie. Katalog jest plikiem zawierającym tabelę z nazwami znajdujących się w nim plików. Poszczególnym numerom i-węzłów w tej tabeli są przyporządkowane nazwy plików. I-węzeł jest plikiem odczytywanym przez jądro systemu w celu uzyskania informacji o każdym pliku. W i-węźle znajdują się dane o uprawnieniach, właścicielu pliku, dacie utworzenia i dacie ostatniego dostępu lub zmiany zawartości pliku, a także fizycznym położeniu bloków danych na dysku zawierającym plik. Dane przechowywane w pliku mogą mieć dowolną postać. Mogą to być dane tekstowe ASCII, binarne lub kombinacja obu form.

Główny system plików

W następującej tabeli przedstawione są najważniejsze części głównego systemu plików `/`, na przykładzie systemu Linux:

Ścieżka	**Zawartość**
`/`	W tym katalogu zwykle nie ma żadnych plików
`/bin`	Pliki wykonywalne poleceń potrzebnych podczas uruchamiania systemu; mogą być używane również przez użytkowników (np. `cp`, `ls`, `mv`)
`/sbin`	Pliki wykonywalne poleceń potrzebnych podczas uruchamiania systemu; nie dla użytkowników
`/etc`	Pliki konfiguracyjne systemu i różnych narzędzi
`/lib`	Biblioteki wspólne dla programów głównego systemu plików
`/root`	Katalog główny użytkownika `root`
`/lib/modules`	Moduły jądra, które mogą być załadowane podczas pracy systemu; dotyczy systemu Linux
`/lost+found`	Pliki lub fragmenty plików uszkodzone w wyniku awarii lub niekontrolowanego zamknięcia systemu
`/dev`	Pliki przeznaczone do obsługi urządzeń
`/tmp`	Katalog plików tymczasowych
`/boot`	W systemie Linux tu są umieszczane pliki loadera LILO i obrazy jądra systemu
`/mnt`	Katalog, w którym różne systemy plików mogą być tymczasowo montowane przez administratora systemu, na przykład `/mnt/dos`,

	`/mnt/ext2`; nie jest to jednak regułą dla wszystkich wariantów Uniksa
`/opt`	Miejsce przechowywania dużych aplikacji

Katalog `/dev`

W katalogu `/dev` znajdują się pliki przeznaczone do obsługi urządzeń. Ponieważ w Uniksie „każdy element jest plikiem", polecenie odczytu danych z urządzenia lub zapis do niego ma tę samą postać, jak operacje wykonywane na zwykłych plikach. Na przykład, polecenie skierowania zawartości pliku do drukarki przyłączonej do portu równoległego ma następującą postać:

```
$ cat plik > /dev/lp0
```

Urządzenia należą do typu blokowego (odczytują lub zapisują dane w blokach np. po 512 bajtów; napędy dyskietek, dyski twarde, napędy CD-ROM) lub znakowego (porty szeregowe i równoległe, myszy).

Oto przykłady urządzeń w katalogu `/dev` (na przykładzie systemu Linux):

Nazwa	**Urządzenie**
`/dev/dsp`	Dedykowany procesor dźwięku
`/dev/fd0`	Pierwszy napęd dyskietek
`/dev/fb0`	Pierwszy bufor ramek (*framebuffer*); warstwa abstrakcji między oprogramowaniem a sprzętem; urządzenie znakowe
`/dev/djs0`	Pierwszy joystick cyfrowy
`/dev/hda`	Dysk główny przyłączony do sterownika głównego
`/dev/hdb`	Dysk podrzędny przyłączony do sterownika głównego
`/dev/hdc`	Dysk główny przyłączony do sterownika pomocniczego
`/dev/hdd`	Dysk podrzędny przyłączony do sterownika pomocniczego
`/dev/ht0`	Pierwszy napęd taśmowy IDE
`/dev/js0`	Pierwszy joystick analogowy
`/dev/lp0`	Pierwszy port równoległy (drukarka, skaner)
`/dev/loop0`	Pierwsza pętla zwrotna
`/dev/md0`	Pierwsza grupa metadysków (RAID)
`/dev/mixer`	Urządzenie przyporządkowane programowi sterującemu dźwiękiem *OSS driver*
`/dev/null`	Urządzenie pełniące rolę pochłaniacza danych; na przykład, jeżeli na standardowym wyjściu nie powinny pojawiać się żadne dane, wygodnym rozwiązaniem jest przekierowanie wyjścia do `/dev/null`
`/dev/psaux`	Połączenie typu PS/2 (mysz, klawiatura)
`/dev/pda`	Dyski IDE przyłączone do portu równoległego
`/dev/pcd0`	Napędy CD-ROM przyłączony do portu równoległego
`/dev/pt0`	Napędy taśmowe przyłączone do portu równoległego
`/dev/parport0`	Porty równoległe „w stanie surowym"; większość urządzeń przyłączonych do portów równoległych ma własne programy sterujące; to urządzenie służy do bezpośredniej komunikacji z portem równoległym
`/dev/random` lub `/dev/urandom`	Generatory liczb losowych jądra systemu; `/dev/random` generuje liczby, wykorzystując informację o entropii sprzętu zainstalowanego w systemie; jeżeli zabraknie danych o entropii układu, kolejna liczba

	losowa jest generowana dopiero po uzyskaniu informacji o entropii; `/dev/urandom` działa na podobnej zasadzie; jednak w sytuacji, kiedy brakuje danych do wyznaczenia entropii układu, `/dev/urandom` generuje liczby pseudolosowe; nie jest to najlepsze rozwiązanie w zastosowaniu do generowania pary kluczy szyfrujących; pierwszy algorytm jest bezpieczniejszy, drugi – szybszy
`/dev/sda`	Pierwsze urządzenie SCSI (dysk twardy, zewnętrzne pamięci masowe)
`/dev/scd`	Pierwszy napęd SCSI CD-ROM
`/dev/ttyS0`	Pierwszy port szeregowy; używany przez mysz, modem, kabel typu null-modem
`/dev/usb`	Urządzenia dołączane do portu USB
`/dev/zero`	Operacja odczytu z tego urządzenia zwraca znak 0; może być przydatne na przykład w celu utworzenia pliku o określonej długości

Urządzenia charakteryzowane są także numerem głównym i numerem pobocznym. Na przykład, urządzenia `hda` i `hdb` mają numer główny 3, co oznacza dysk twardy. Numer poboczny zmienia się w zależności od partycji. Definicje numerów głównych i pobocznych dla systemu Linux są zwykle zapisane w pliku `/usr/src/linux/include/linux/major.h`. Numery główne i poboczne są objaśnione również w pliku `devices.txt` w katalogu `/usr/src/linux/Documentation`. Prawie wszystkie urządzenia są utworzone podczas instalacji systemu. Administrator może jednak utworzyć nowy plik urządzenia za pomocą polecenia `mknod` lub skryptu `/dev/MAKEDEV`.

Katalog `/etc`

W katalogu `/etc` i jego podkatalogach znajdują się pliki konfiguracyjne systemu. Znaczenie wielu z nich wyjaśnia następująca tabela:

Plik	**Zawartość**
`/etc/X11`	Gałąź katalogów zawierająca wszystkie pliki konfiguracyjne interfejsu graficznego X Window
`/etc/adjtime`	Parametry operacji uaktualniania zegara systemu
`/etc/aliases`	Lista odbiorców poczty adresowanej do pseudo-użytkowników (na przykład `bin`, `daemon`, `adm`, `lp`, `sync`, `shutdown`, `halt`, `mail`, `news`, `operator`)
`/etc/at.deny`	Lista użytkowników z zablokowanym dostępem do demona `at`; polecenie `at` umożliwia wykonywanie programów w czasie wyznaczonym przez użytkownika

`/etc/cron.d`
`/etc/cron.daily`
`/etc/cron.monthly`
	Katalogi zawierające skrypty wykonywane regularnie przez demona `cron`

`/etc/crontab`	Plik konfiguracyjny demona `cron`; administrator może definiować oddzielne tablice konfiguracyjne dla poszczególnych użytkowników
`/etc/csh.login`	Plik konfiguracyjny `.login` powłoki C dla całego systemu
`/etc/csh.logout`	Plik `.logout` powłoki C dla całego systemu; odczytywany przy wylogowaniu użytkownika
`/etc/csh.cshrc`	Plik konfiguracyjny `.cshrc` powłoki C dla całego systemu;

	odczytywany przy każdym uruchomieniu powłoki C
`/etc/rc` lub	Skrypty lub katalogi skryptów uruchamianych podczas startu
`/etc/rc.d` lub	systemu lub podczas zmiany trybu działania systemu
`/etc/rc?.d`	
`/etc/passwd`	Plik zawierający nazwy użytkowników, imiona i nazwiska, zaszyfrowane hasła
`/etc/fdprm`	Tabela parametrów dyskietek
`/etc/fstab`	Lista systemów plików montowanych automatycznie poleceniem `mount -a`
`/etc/group`	To samo co plik `passwd`, ale w odniesieniu do grup
`/etc/inittab`	Plik konfiguracyjny polecenia `init`
`/etc/issue`	Krótki komunikat przed logowaniem
`/etc/login.defs`	Konfiguracja polecenia `login`
`/etc/magic`	Opis różnych formatów plików wykorzystywanych przez polecenie `file`
`/etc/motd`	*Message of the day*; komunikat wyświetlany zaraz po zalogowaniu użytkownika
`/etc/mtab`	Lista zamontowanych systemów plików; uaktualniana automatycznie podczas wykonywania polecenia `mount`; informację o zamontowanych systemach plików można uzyskać również za pomocą polecenia `df`
`/etc/shadow`	Szyfrowany plik haseł dostępny tylko dla administratora
`/etc/printcap`	Dane dotyczące różnych typów drukarek; umożliwia ujednoliconą obsługę różnych drukarek
`/etc/profile`	Pliki konfiguracyjne powłoki Bourne'a, Korna lub C;
`/etc/csh.login`	odczytywane podczas uruchamiania systemu
`/etc/csh.cshrc`	
`/etc/securetty`	Definicje terminali „bezpiecznych”, tj. takich, z których administrator może logować się do systemu; zwykle obejmują one jedynie wirtualne konsole, co utrudnia włamania na konto `root` z sieci lub przez połączenie modemowe; logowania z sieci powinny być blokowane; w celu wykonania czynności administracyjnych zaleca się logowanie na konto nieuprzywilejowane, a następnie wykonanie polecenia `su` lub `sudo`, aby uzyskać prawa administratora
`/etc/shells`	Lista dozwolonych powłok; polecenie `chsh` (o ile nie zostało zablokowane przez administratora) umożliwia zmianę domyślnej powłoki użytkownika na inną z tej listy; `ftpd`, serwer usługi `ftp`, odmawia dostępu użytkownikowi próbującemu połączyć się z serwerem, jeśli powłoka użytkownika nie figuruje na liście
`/etc/termcap`	Dane dotyczące różnych typów terminali; umożliwia ujednoliconą obsługę różnych terminali

Katalog `/lib`

Katalog `/lib` zawiera moduły jądra Linuksa i obrazy bibliotek wspólnych niezbędnych do uruchomienia systemu i obsługi plików wykonywalnych w katalogach `/bin` i `/sbin`.

Katalog `/proc`

Katalog `/proc` jest wirtualnym systemem plików, istniejącym jedynie w pamięci komputera. Zawiera bieżące dane dotyczące pracy systemu operacyjnego. Dane każdego działającego procesu są dostępne w podkatalogu, którego nazwa jest numerem procesu PID. Dostępne są również informacje o konfiguracji jądra systemu i konfiguracji sprzętu.

Katalog `/usr`

Katalog `/usr` zawiera programy dostępne z reguły dla wszystkich użytkowników systemu. Administrator może niektóre aplikacje, biblioteki i dokumentację umieścić w innych gałęziach systemu, na przykład w `/usr/local`.

Ścieżka	**Zawartość**
`/usr/X11R6`	Wszystkie pliki interfejsu graficznego X Window
`/usr/bin`	Pliki wykonywalne poleceń dostępnych dla wszystkich użytkowników systemu
`/usr/doc` lub `/usr/share/doc`	Główny katalog przeznaczony do przechowywania dokumentacji systemu
`/usr/include`	Pliki nagłówkowe języka C (z rozszerzeniem `.h`)
`/usr/lib`	Biblioteki dla programów gałęzi `/usr`; niektóre pliki konfiguracyjne dla całego systemu
`/usr/local`	Katalog, w którym umieszczone są programy i inne pliki zainstalowane przez administratora w lokalnym systemie
`/usr/sbin`	Polecenia administracyjne systemu, które nie muszą znajdować się w `/sbin` w głównym systemie plików; większość programów typu serwer znajduje się w tym katalogu
`/usr/share/man`	Strony pomocy `man`
`/usr/src`	Źródła jądra Linuksa, pliki nagłówkowe i dokumentacja

Katalog `/var`

W katalogu `/var` znajdują się pliki o zmieniającej się zawartości. Tu są umieszczone pliki dzienników, kolejek drukowania, poczty (zwykle w katalogu `/var/spool/mail`) i dane tymczasowe.

Logowanie użytkownika do systemu

Po nawiązaniu połączenia z systemem, użytkownik musi podać nazwę użytkownika i hasło. Nazwa użytkownika jednoznacznie identyfikuje go w systemie. Oczywiście, hasła są tajne i każdy użytkownik powinien znać jedynie swoje hasło.

Uwaga: w Uniksie wielkie i małe litery są rozróżniane. Dlatego nazwa użytkownika i hasło powinny być wpisywane dokładnie w takiej postaci, w jakiej zostały początkowo podane. Nazwa użytkownika najczęściej składa się z małych liter.

Typ terminala

Większość komputerów działa prawidłowo, jeśli zdefiniować terminal typu `vt100`. Innymi często spotykanymi wariantami terminala są `sun` (na stacjach roboczych firmy Sun) lub `xterm` (na komputerach z uruchomioną obsługą X-Windows).

Typ terminala jest zbiorem definicji określających sposób komunikowania się z otwartą sesją użytkownika.

Oto przykład ustawienia definicji terminala:

```
setenv TERM vt100              w powłoce C
TERM=vt100; export TERM        w powłoce Bourne'a
```

Hasła

Po utworzeniu nowego konta użytkownik otrzymuje początkowe hasło, które powinien później zmienić tak, by było znane jedynie jemu samemu. Do zmiany hasła służy polecenie `passwd`. Użytkownik musi wpisać bieżącą postać hasła, a następnie dwukrotnie wpisać nowe hasło. Jeżeli bieżące hasło zostanie podane niepoprawnie lub nowe hasło nie zostanie wpisane dwukrotnie w tej samej postaci, hasło pozostanie bez zmian.

Administrator systemu może zainstalować program sprawdzający odporność hasła na złamanie. Jeżeli hasło wybrane przez użytkownika nie spełnia pewnych minimalnych kryteriów, system nie zaakceptuje zmiany hasła. Hasło nie powinno być zapisywane ani udostępniane innym osobom.

Oto podstawowe zalecenia, na które należy zwrócić uwagę podczas wyboru hasła:

Tak	**Nie**
Użyj różnych znaków (liter, cyfr, znaków specjalnych)	Słowa jakiegokolwiek języka
Użyj liter wielkich i małych	Nazwy własne
Hasło powinno składać się z co najmniej 6 znaków	Dane, jakie mogą znajdować się w portfelu lub w notesie użytkownika
Wybierz hasło, które jesteś w stanie zapamiętać	Dane o użytkowniku, które są powszechnie znane, np. znaki z rejestracji samochodu, imię kota lub psa, itd.
Zmieniaj hasło stosunkowo często	Znaki sterujące; niektóre systemy nie potrafią

ich obsłużyć

Upewnij się, czy nikt nie obserwuje Cię podczas wpisywania hasła.

Wylogowanie z systemu

Naciśnięcie kombinacji klawiszy **Ctrl+D** oznacza koniec strumienia danych i powoduje wylogowanie użytkownika, o ile ta opcja nie została wyłączona przez administratora systemu. Kombinacja klawiszy **Ctrl+C** powoduje przerwanie bieżącego procesu. Polecenie `logout` powoduje wyjście z systemu, a polecenie `exit` – wyjście z bieżącej powłoki.

Identyfikacja użytkownika

Użytkownik jest identyfikowany za pomocą numerów użytkownika i grupy (`userid` i `groupid`). Zwykle znajomość tych liczbowych identyfikatorów nie jest konieczna, ponieważ nazwa użytkownika jednoznacznie odpowiada identyfikatorowi `userid`, a nazwa grupy odpowiada identyfikatorowi `groupid`. Użytkownik może należeć do wielu grup. Główną grupą użytkownika jest grupa, której przyporządkowana jest nazwa użytkownika w systemowym pliku haseł.

Aby uzyskać informacje o identyfikatorze `userid`, wykonaj polecenie `id`. Wynik polecenia ma następującą postać:

```
$ id
uid=1101(piotr) gid=513(Brak) groups=513(Brak)
```

Wyświetlane mogą też być inne grupy, do których użytkownik należy, na przykład

```
$ id
uid=1101(piotr) gid=5(operator),14(sysadmin),110(uts)
```

Aby uzyskać informacje o wszystkich grupach, do których należy użytkownik, wykonaj następujące polecenie:

```
$ groups
sysadmin uts operator
```

Ogólna postać poleceń Uniksa

Polecenie w systemie Unix ma następującą postać:

`polecenie` [*`opcje`*] [*`argumenty`*]

Argumentem jest zwykle plik lub grupa plików. Opcje służą do modyfikowania sposobu wykonania polecenia.

W poleceniach wielkie i małe litery są rozróżniane. Dlatego *`polecenie`* i *`Polecenie`* nie są identyczne.

Opcje są zwykle poprzedzone łącznikiem (znakiem –) i mogą być zapisywane łącznie w następujący sposób:

polecenie -[*opcja*][*opcja*][*opcja*]

Na przykład, polecenie

```
ls -alr
```

wyświetla informacje o wszystkich plikach znajdujących się w katalogu bieżącym. Poza nielicznymi wyjątkami, opcje mogą być również zapisywane osobno, na przykład:

```
ls -a -l -r
```

Czasem opcje wymagają podania parametrów i wtedy są pisane najczęściej osobno.

Zdarzają się wyjątki od tych reguł. W niektórych poleceniach nie ma łącznika (znak –) przed opcjami, a w innych nie można grupować opcji (każda opcja musi być poprzedzona łącznikiem i oddzielona odstępem od pozostałych składników polecenia).

Opis składni poszczególnych poleceń i dostępnych opcji jest zawarty w plikach pomocy `man`.

Pliki pomocy `man`

Za pomocą polecenia `man` można wyświetlić opis poszczególnych poleceń. Oto przykład:

```
$ man passwd

PASSWD(1)                    User utilities                    PASSWD(1)

NAME
       passwd - update a user's authentication tokens(s)

SYNOPSIS
       passwd [-k] [-l] [-u [-f]] [-d] [-S] [username]

DESCRIPTION
       Passwd is used to update a user's authentication token(s).

       Passwd is configured to work through the Linux-PAM API. Essentially,
       it initializes itself as a "passwd" service with Linux-PAM and
       utilizes configured password modules to authenticate and then update
       a user's password.

       A simple entry in the Linux-PAM configuration file for this service
       would be:

        #
        # passwd service entry that does strength checking of
        # a proposed password before updating it.
        #
        passwd password requisite \
                    /usr/lib/security/pam_cracklib.so retry=3
        passwd password required \
                    /usr/lib/security/pam_unix.so use_authtok
```

```
        #

       Note, other module-types are not required for this application to
       function correctly.

OPTIONS
       -k   The option, -k, is used to indicate that the update should only
            be for expired authentication tokens (passwords); the user
            wishes to keep their non-expired tokens as before.

       -l   This option is used to lock the specified account and it is
            available to root only. The locking is performed by rendering
            the encrypted password into an invalid string (by prefixing the
            encrypted string with an !).

       --stdin
            This option is used to indicate that passwd should read the
            new password from standard input, which can be a pipe.

       -u   This is the reverse of the -l option - it will unlock the
            account password by removing the ! prefix. This option is
            available to root only. By default passwd will refuse to create
            a passwordless account (it will not unlock an account that has
            only "!" as a password). The force option -f will override
            this protection.

       -d   This is a quick way to disable a password for an account. It
            will set the named account passwordless. Available to root
            only.

       -n   This will set the minimum password lifetime, in days, if the
            user's account supports password lifetimes. Available to root
            only.

       -x   This will set the maximum password lifetime, in days, if the
            user's account supports password lifetimes. Available to  root
            only.

       -w   This will set the number of days in advance the user will begin
            receiving warnings that her password will expire, if the user's
            account supports password lifetimes. Available to root only.

       -i   This will set the number of days which will pass before an
            expired password for this account will be taken to mean that
            the account is inactive and should be disabled, if the user's
            account supports password lifetimes. Available to root only.

       -S   This will output a short information about the status of the
            password for a given account. Available to root user only.
```

Opis tego polecenia jest dłuższy i został przytoczony jedynie w skróconej postaci. Aby przewijać tekst do przodu lub wstecz, należy używać tych samych klawiszy, co podczas przeglądania pliku za pomocą polecenia `less` lub `more`. Naciśnięcie spacji powoduje przesunięcie tekstu o jeden ekran do przodu. Klawisz **b** przewija tekst o jeden ekran wstecz. Naciśnięcie klawisza **Enter** przesuwa tekst jeden wiersz do przodu. Aby zakończyć przeglądanie tekstu, należy nacisnąć klawisz **q**.

Definicje parametrów terminala

Polecenie `stty` służy do uzyskiwania informacji o bieżącej konfiguracji terminala i dokonywania zmian jego parametrów. Za pomocą tego polecenia można szczegółowo konfigurować parametry wejścia i wyjścia. Dla niezaawansowanych najważniejszą opcją jest wybór klawisza usuwającego znaki. Oto niektóre inne opcje:

Kombinacja klawiszy powodująca usunięcie całego wiersza
Szybkość transmisji danych
Sprawdzanie parzystości podczas transmisji danych
Sprzętowe sterowanie przepływem (ang. *hardware flow control*)
Sposób interpretowania klawisza **Tab**
Mapowanie wielkich liter na małe

Sposób działania `stty` w znacznym stopniu zależy od szczegółów systemu operacyjnego i sprzętu. Oto niektóre opcje tego polecenia:

Opcja	**Znaczenie**
(brak)	Drukowanie ustawień terminala
`all` lub `-a`	Drukowanie informacji o wszystkich dostępnych opcjach
`eof`	Zakończenie strumienia danych (zwykle **Ctrl+D**)
`kill`	Definiowanie kombinacji klawiszy usuwającej cały wiersz (zwykle **Ctrl+U**)
`erase`	Przyporządkowanie klawisza ERASE (zwykle **Backspace** lub **Delete**)
`intr`	Przyporządkowanie klawisza INTERRUPT (domyślnie **Ctrl+C**)

Zapis **Ctrl+D** oznacza, że należy nacisnąć klawisz **Ctrl** i przytrzymać go, po czym nacisnąć klawisz **D**. W przypadku klawiszy sterujących wielkość liter nie odgrywa roli. Na przykład, kombinacja **Ctrl+C** jest równoważna **Ctrl+c**.

Zdarza się, że funkcja usuwania znaków jest przyporządkowana klawiszowi **Delete** (w `stty` oznaczonym `^?`). Aby nadać klawiszowi **Backspace** funkcję usuwania znaków, należy wykonać następujące polecenie:

```
$ stty erase ^H
```

Po wpisaniu `stty erase` i naciśnięciu spacji należy nacisnąć klawisz **Backspace**. W ten sam sposób można przyporządkować funkcję usuwania znaków klawiszowi **Delete**. Nowe definicje obowiązują jedynie w bieżącej sesji. Jeżeli mają obowiązywać przy każdym zalogowaniu, polecenie należy umieścić w pliku definiującym środowisko danej powłoki. Aby prawidłowo wpisać `^H` w edytorze `vi`, należy w trybie wpisywania tekstu nacisnąć kombinację klawiszy **Ctrl+V**, a następnie nacisnąć **Ctrl+H**.

Oto przykład ustawień terminala w systemie Red Hat Linux 8.0:

```
$ stty -a
speed 38400 baud; rows 25; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon
-ixoff
-iuclc -ixany -imaxbel
```

```
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0
ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt
echoctl echoke
```

Operacje dotyczące katalogów

Każdy użytkownik ma własny katalog główny i zarządza wyrastającym z niego własnym drzewem katalogów. Podstawowe operacje na katalogach są przedstawione w następującej tabeli:

Polecenie	**Znaczenie**
`cd`	Przejście do innego katalogu
`ls`	Drukowanie zawartości katalogu
`mkdir`	Utworzenie nowego katalogu
`pwd`	Drukowanie pełnej ścieżki bieżącego katalogu
`rmdir`	Usunięcie katalogu

`cd`

Do poruszania się po drzewie katalogów służy polecenie `cd`, w którym jako argument można podać zarówno ścieżkę pełną, jak i niepełną:

Polecenie	**Znaczenie**
`cd`	Przejście do katalogu głównego bieżącego użytkownika
`cd /`	Przejście do głównego katalogu systemowego
`cd ..`	Przejście w górę o jeden poziom
`cd ../..`	Przejście w górę o dwa poziomy
`cd /usr/local`	Przejście z podaniem ścieżki pełnej
`cd kat1/kat2`	Przejście z podaniem ścieżki niepełnej
`cd ~adam/kat1`	Przejście do podkatalogu `kat1`, znajdującego się w katalogu głównym użytkownika `adam`

`ls`

Zawartość katalogu jest wyświetlana za pomocą polecenia `ls`. Oto pliki znajdujące się w katalogu `/bin` (w systemie Linux Red Hat 8.0):

```
$ ls /bin
arch           df             hostname  nice           su
ash            dmesg          igawk     nisdomainname  sync
ash.static     dnsdomainname  ipcalc    pgawk          tar
aumix-minimal  doexec         kbd_mode  ping           tcsh
awk            domainname     kill      ps             touch
basename       dumpkeys       link      pwd            true
bash           echo           ln        red            umount
bash2          ed             loadkeys  rm             uname
bsh            egrep          login     rmdir          unicode_start
cat            env            ls        rpm            unicode_stop
chgrp          ex             mail      rvi            unlink
chmod          false          mkdir     rview          usleep
chown          fgrep          mknod     sed            vi
cp             gawk           mktemp    setfont        view
cpio           gettext        more      setserial      ypdomainname
csh            grep           mount     sh             zcat
cut            gtar           mt        sleep
```

```
date            gunzip          mv          sort
dd              gzip            netstat     stty
```

Za pomocą opcji `-l` można uzyskać informacje o rozmiarach plików, uprawnieniach, dacie utworzenia, dacie ostatniej modyfikacji i dacie ostatniego dostępu do pliku. Oto zawartość katalogu użytego w poprzednim przykładzie (aby nie zajmować zbyt wiele miejsca w książce, wydruk został ograniczony do pierwszych kilkunastu pozycji katalogu):

```
$ ls -l
total 7072
-rwxr-xr-x    1 root     root          4330 Aug 30  2002 arch
-rwxr-xr-x    1 root     root        110048 Jul 18  2002 ash
-rwxr-xr-x    1 root     root        505685 Jul 18  2002 ash.static
-rwxr-xr-x    1 root     root         10296 Aug  4  2002 aumix-mini-
mal
lrwxrwxrwx    1 root     root             4 Mar 12 21:11 awk -> gawk
-rwxr-xr-x    1 root     root         10680 Aug 29  2002 basename
-rwxr-xr-x    1 root     root        626188 Aug 23  2002 bash
lrwxrwxrwx    1 root     root             4 Mar 12 21:08 bash2 ->
bash
lrwxrwxrwx    1 root     root             3 Mar 12 21:23 bsh -> ash
-rwxr-xr-x    1 root     root         19154 Jul  1  2002 cat
-rwxr-xr-x    1 root     root         18136 Sep  2  2002 chgrp
-rwxr-xr-x    1 root     root         18072 Sep  2  2002 chmod
-rwxr-xr-x    1 root     root         20120 Sep  2  2002 chown
-rwxr-xr-x    1 root     root         49548 Sep  2  2002 cp
```

Pierwszy znak każdego wiersza oznacza rodzaj obiektu:

Znak	**Typ obiektu**
`d`	katalog
`-`	zwykły plik
`b`	plik specjalny typu blokowego
`c`	plik specjalny typu znakowego
`l`	dowiązanie symboliczne
`s`	gniazdo (ang. *socket*)

Kolejne 9 znaków dzieli się na 3 grupy po 3 znaki. Oznaczają one uprawnienia dostępu. Pierwsze trzy znaki informują o prawach użytkownika, następne trzy – o prawach grupy, a pozostałe – o prawach wszystkich pozostałych użytkowników systemu.

Uprawnienia są oznaczone następującymi znakami:

Znak	**Rodzaj uprawnienia**
`r`	Prawo odczytu
`w`	Prawo zapisu
`x`	prawo uruchamiania
`-`	brak uprawnień

W następującej tabeli podane są najczęściej używane opcje polecenia `ls`.

Opcja	**Znaczenie**
`-1`	Drukowanie tylko jednej nazwy pliku w wierszu
`-a`	Drukowanie wszystkich plików, również tych, których nazwa rozpoczyna się kropką
`-A`	Drukowanie wszystkich nazw, oprócz kropki `.` (symbol katalogu bieżącego) i dwóch kropek `..` (symbol katalogu nadrzędnego)
`-d`	Drukowanie tylko nazw katalogów
`-F`	Do nazw dołączone są następujące symbole oznaczające typ obiektu: `/` katalog `=` gniazdo `@` dowiązanie symboliczne `*` plik wykonywalny
`-l`	Drukowanie szczegółowych informacji: praw dostępu, właściciela, rozmiaru, czasu ostatniej modyfikacji. Jeżeli plik jest dowiązaniem symbolicznym, ścieżka do obiektu, na który wskazuje dowiązanie, jest poprzedzona symbolem `-->`
`-L`	Jeżeli plik jest dowiązaniem symbolicznym, powoduje wyświetlenie informacji o pliku lub katalogu, na który wskazuje dowiązanie, a nie o samym dowiązaniu
`-o`	Drukowanie tych samych informacji, co w przypadku opcji `-l`, oprócz nazwy grupy
`-R`	Rekurencyjne drukowanie zawartości podkatalogów
`-r`	Drukowanie plików w odwrotnej kolejności
`-s`	Drukowanie rozmiaru pliku w blokach
`-S`	Sortowanie według rozmiaru plików
`-t`	Drukowanie według czasu ostatniej modyfikacji, od najnowszych do najstarszych
`-u`	Drukowanie według czasu ostatniego dostępu

Przykłady

Drukowanie katalogu z uwzględnieniem nazw plików i katalogów rozpoczynających się kropką:

```
$ ls -a
```

Pełna informacja o całej zawartości bieżącego katalogu i jego podkatalogów:

```
$ ls -laR
```

Drukowanie plików uporządkowanych względem czasu ostatniej modyfikacji, od najstarszych do najnowszych:

```
$ ls -latr
```

Drukowanie plików uporządkowanych względem czasu ostatniego dostępu, od najstarszych do najnowszych:

```
$ ls -latru
```

Drukowanie pełnej informacji o plikach uporządkowanych względem rozmiaru, od największych do najmniejszych:

```
$ ls -lS
```

Drukowanie pełnej informacji o plikach uporządkowanych względem rozmiaru, od najmniejszych do największych:

```
$ ls -lSr
```

mkdir

Katalogi są tworzone za pomocą polecenia `mkdir`. Oto najczęściej używane opcje:

Opcja	Znaczenie
`-p`	Utworzenie katalogów pośrednich, jeśli to konieczne
`-m` *`tryb`*	Ustawienie uprawnień dostępu

Argument tego polecenia można podać zarówno w postaci pełnej ścieżki, jak i niepełnej.

```
$ mkdir /home/adam/dane
```

Do wykonania tej samej operacji w katalogu `/home/adam` wystarczy podać następujące polecenie:

```
$ mkdir dane
```

pwd

Polecenie `pwd` drukuje ścieżkę do bieżącego katalogu (ang. *path to working directory*), na przykład:

```
$ pwd
/home/adam/programy
```

rmdir

Polecenie `rmdir` usuwa katalog. Usunąć można jedynie katalog pusty:

```
$ rmdir katalog
```

Na przykład,

```
$ rmdir dane
```

lub

```
$ rmdir /home/adam/dane
```

Jezeli katalog zawiera pliki lub podkatalogi, należy najpierw usunąć jego zawartość lub przenieść ją w inne miejsce. W powłoce Bourne'a można jednak usunąć katalog wraz z całą zawartością za pomocą polecenia `rm -r`:

```
$ rm -r katalog
```

Innym ograniczeniem jest niemożność usunięcia katalogu bieżącego (oznaczanego kropką .).

Podstawowe operacje na plikach

W tabeli podane są polecenia tworzenia, kopiowania, usuwania i zmiany uprawnień plików.

Polecenie	Znaczenie
`chmod` *plik*	Zmiana uprawnień dostępu do pliku lub katalogu
`chown` *właściciel plik*	Zmiana właściciela pliku; tylko administrator ma prawo to zrobić
`cp` *plik1 plik2*	Kopiowanie *plik1* na *plik2*
`mv` *plik1 plik2*	Zmiana nazwy pliku z *plik1* na *plik2*
`rm` *plik*	Usunięcie pliku lub katalogu
`umask` [*tryb*]	Ustawienie domyślnych uprawnień do nowo tworzonych plików lub wyświetlenie informacji o bieżącym ustawieniu (polecenie `umask` bez argumentu)

Uprawnienia

Aby odczytać uprawnienia dostępu do plików, należy wykonać polecenie `ls -l`, które zostało omówione w poprzednim rozdziale. Na przykład, uprawnienia `-rwxr-xr-x` oznaczają, że właściciel pliku ma pełnię praw, a grupa i wszyscy inni mają prawo odczytu i uruchamiania pliku. Każdy rodzaj uprawnienia ma przyporządkowaną wartość liczbową lub symbol:

Rodzaj uprawnienia	Wartość	Symbol
Odczyt	4	`R`
Zapis	2	`W`
uruchamianie	1	`X`

Wartości uprawnień sumują się. Na przykład, 6 oznacza prawo odczytu i zapisu, a 7 – prawo odczytu, zapisu i wykonywania.

chmod

Polecenie zmiany praw dostępu `chmod` może mieć kilka różnych postaci. Na przykład,

```
$ chmod 700 plik
```

oznacza pozostawienie właścicielowi pliku pełni praw i odebranie wszystkich uprawnień grupie oraz pozostałym użytkownikom. To samo można zapisać w następującej postaci:

```
$ chmod u=rwx,go= plik
```

Litera `u` oznacza uprawnienia użytkownika, `g` – uprawnienia grupy, `o` – uprawnienia pozostałych użytkowników, `a` – uprawnienia wszystkich, łącznie z właścicielem pliku.

Polecenie

```
$ chmod 644 plik
```

przyznaje prawa odczytu i zapisu właścicielowi oraz prawo odczytu grupie i pozostałym użytkownikom. Oto równoważna postać ostatniego polecenia:

```
$ chmod u=rw, go=r plik
```

Aby dodać uprawnienia lub je zmniejszyć, należy użyć znaku + lub −. W tym przykładzie użytkownik otrzymuje prawo wykonywania pliku, a grupa i wszyscy inni pozbawieni są prawa czytania i uruchamiania:

```
$ chmod u+=x, go-=wx plik
```

umask

Aby sprawdzić domyślne ustawienia dotyczące uprawnień do nowo tworzonych plików, należy wykonać polecenie `umask`, na przykład

```
$ umask
022
```

Odpowiedź `022` oznacza, że właściciel ma pełne prawa do nowo utworzonego pliku lub katalogu, a grupa i pozostali mają prawo odczytu i wykonywania (jeśli jest to plik wykonywalny), ale nie mają prawa zapisu do pliku lub katalogu. Następujące polecenie odbiera grupie i pozostałym użytkownikom wszystkie prawa dostępu do nowo tworzonych plików:

```
$ umask 077
```

Odpowiednią postać polecenia `umask` warto zapisać w pliku `.profile` (dla powłoki Bourne'a) lub innym pliku konfigurującym środowisko użytkownika powłoki.

chown

Właściciela pliku można zmienić za pomocą polecenia `chown`. W większości wariantów Uniksa prawo to przysługuje jedynie administratorowi. Oto przykład:

```
$ chown nowy_uzytkownik plik
```

Za pomocą opcji `-R` można tę operację wykonać rekursywnie na całym drzewie katalogowym.

cp

Polecenie `cp` kopiuje pliki lub katalogi. Za pomocą opcji `-r` można rekursywnie skopiować całą zawartość katalogu. Aby uniknąć przypadkowego usunięcia istniejących już plików, można użyć opcji `-i`. Wówczas użytkownik musi potwierdzać operację kopiowania, naciskając klawisz **y**.

Przykład. Aby skopiować całą zawartość bieżącego katalogu, do innego, na przykład do `/home/ola/dane`, należy wykonać następujące polecenie:

```
$ cp ./* /home/ola/dane
```

W miejsce `./*` można, oczywiście, podać pełną ścieżkę dostępu do katalogu, na przykład `/home/ola/programy/dane/*`

Znak `*` zastępuje dowolny ciąg znaków. Wyrażenie `./*` oznacza wszystkie pliki z bieżącego katalogu.

mv

Składnia polecenia `mv` jest bardzo podobna do polecenia `cp`. Polecenie przeniesienia plików z poprzedniego przykładu miałoby następującą postać:

```
$ mv ./* /home/ola/dane
```

Podobnie, jak w przypadku polecenia `cp`, opcja `-i` powoduje wyświetlenie pytania o potwierdzenie operacji. Czasem przydatna jest opcja `-f`, anulująca opcję `-i` (o ile została wcześniej zdefiniowana). Jeśli *`plik2`* istnieje, polecenie

```
$ mv -f plik1 plik2
```

ignoruje ten fakt i wymusza zapisanie w to miejsce kopii pliku *`plik1`*. Trzeba pamiętać o tym, że to polecenie zmienia jedynie zapis w tablicy zawartości katalogu, a sam plik pozostaje w tym samym miejscu na dysku. Jeżeli jednak *`plik2`* ma być dostępny w katalogu należącym do innej partycji, zostaje przeniesiony fizycznie w nowe miejsce. Na przykład, jeżeli katalogi `/usr/bin` i `/usr/local/bin` są ulokowane na różnych partycjach, to polecenie przeniesienia pliku wykonywalnego przeglądarki internetowej,

```
$ cp /usr/bin/netscape /usr/local/bin/netscape
```

zmienia także fizyczne położenie pliku. Aby rekurencyjnie skopiować katalog razem z podkatalogami, należy użyć opcji `-r`.

rm

Pliki są usuwane poleceniem `rm`. Aby wyświetlić pytanie o potwierdzenie tej operacji, należy użyć opcji `-i`. Następujące polecenie usuwa w trybie interakcyjnym wszystkie pliki z bieżącego katalogu:

```
$ rm -i ./*
```

Do rekursywnego usuwania całej gałęzi drzewa katalogowego (wszystkich podkatalogów i zawartych w nich plików) należy użyć opcji `-r`, na przykład:

```
$ rm -r kat1
```

W tym przykładzie usuwana jest cała zawartość katalogu *`kat1`*. Polecenie `rm` usuwa z katalogu wpis odnoszący się do usuwanego pliku. Zawartość pliku pozostaje jeszcze na dysku

w nienaruszonym stanie. System nie potrafi już jednak połączyć bloków danych z nazwą pliku i nie ma żadnego polecenia, które mogłoby przywrócić plik. Początkujący użytkownicy Uniksa zmieniają czasem domyślną postać poleceń `cp`, `mv` i `rm`, dopisując do każdego z nich opcję `-i` (definiując odpowiednią funkcję w pliku `.profile` dla powłoki Bourne'a lub `alias` w pliku `.cshrc` dla powłoki C).

Wyświetlanie zawartości pliku

Podstawowe polecenia, służące do wyświetlania treści pliku są zawarte w następującej tabeli:

Polecenie	Znaczenie
`cat` *plik*	Wyświetlenie całej zawartości pliku
`head` *plik*	Wyświetlenie początkowych wierszy pliku (domyślnie 10)
`more` *plik*	Wyświetlanie pliku fragmentami mieszczącymi się w oknie terminala
`less` *plik*	jw.
`od` *plik*	Drukowanie zawartości pliku na ekranie w różnych notacjach: ósemkowej, dziesiętnej, zmiennoprzecinkowej, szestnastkowej i znakowej
`tail` *plik*	Drukowanie końcowych wierszy pliku (domyślnie 10)

cat

Do wyświetlania całej zawartości pliku służy polecenie `cat`:

```
$ cat plik
```

Można wyświetlić także treść kilku plików, jeden po drugim:

```
$ cat plik1 plik2 plik3
```

Jest to wygodny sposób na łączenie wielu plików w jedną całość. W tym celu wystarczy skierować wyjście do nowego pliku:

```
$ cat plik1 plik2 plik3 > nowy_plik
```

Użycie opcji `-n` powoduje ponumerowanie wierszy na wyjściu.

head

Polecenie `head` wyświetla początek pliku (domyślnie 10 wierszy):

```
$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
```

```
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
```

Oto przykład polecenia, które wyświetla 30 początkowych wierszy pliku:

```
$ head -30 plik
```

`more` lub `less`

Aby przeglądać zawartość pliku fragmentami mieszczącymi się na ekranie, należy użyć polecenia `more` lub `less`. Polecenie `more` funkcjonuje w komercyjnych wariantach Uniksa. Jego odpowiednikiem w systemach Linux i FreeBSD jest `less`. Polecenie

```
$ more plik
```

wyświetla początek pliku. Aby obejrzeć dowolną część pliku, można przewinąć tekst za pomocą spacji. Jedno naciśnięcie tego klawisza przesuwa tekst o jeden ekran do przodu. Aby cofnąć tekst, należy nacisnąć klawisz **b**. Naciśnięcie klawisza **Enter** przesuwa tekst o jeden wiersz do przodu. W następującej tabeli umieszczone są opisy funkcji najczęściej używanych klawiszy polecenia `more` (lub `less`):

Polecenie	**Znaczenie**
`spacja`	Przesunięcie tekstu jeden ekran do przodu
`Enter`	Przesunięcie tekstu jeden wiersz do przodu
`q`	Zakończenie przeglądania pliku
`h`	Wyświetlenie podręcznej pomocy
`b`	Przesunięcie tekstu jeden ekran wstecz
/slowo	Wyszukanie w pozostałej części tekstu łańcucha *slowo*

`od`

Za pomocą polecenia `od` można wyświetlić zawartość plików w zapisie ósemkowym, znakowym (ASCII), dziesiętnym, zmiennoprzecinkowym lub szesnastkowym.

Opcja	**Znaczenie**
`-a`	Wyświetlenie bajtów w postaci znaków ASCII lub nazw znaków
`-b`	Wyświetlenie bajtów w notacji ósemkowej
`-c`	Wyświetlenie bajtów w postaci znaków ASCII

Powiedzmy, że plik zawiera zbiór wszystkich znaków ASCII. Próbując wyświetlić go za pomocą polecenia `cat` otrzymamy następujący wynik:

```
$ cat znaki_ASCII
☺☻♥♦♣♠
♫♀►◄↕‼¶§▬↨↑↓→←▲▼ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~⌂
```

Pierwsze 32 znaki oraz ostatni znak w tym zbiorze są niedrukowalne. Oto ten sam plik wyświetlony trzema innymi sposobami:

```
$ od -a znaki_ASCII
0000000 nul soh stx etx eot enq ack bel  bs  ht  nl  vt  ff  cr  so  si
0000020 dle dc1 dc2 dc3 dc4 nak syn etb can  em sub esc  fs  gs  rs  us
0000040  sp   !   "   #   $   %   &   '   (   )   *   +   ,   -   .   /
0000060   0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ?
0000100   @   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
0000120   P   Q   R   S   T   U   V   W   X   Y   Z   [   \   ]   ^   _
0000140   `   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o
0000160   p   q   r   s   t   u   v   w   x   y   z   {   |   }   ~ del
0000200
```

```
$ od -b znaki_ASCII
0000000 000 001 002 003 004 005 006 007 010 011 012 013 014 015 016 017
0000020 020 021 022 023 024 025 026 027 030 031 032 033 034 035 036 037
0000040 040 041 042 043 044 045 046 047 050 051 052 053 054 055 056 057
0000060 060 061 062 063 064 065 066 067 070 071 072 073 074 075 076 077
0000100 100 101 102 103 104 105 106 107 110 111 112 113 114 115 116 117
0000120 120 121 122 123 124 125 126 127 130 131 132 133 134 135 136 137
0000140 140 141 142 143 144 145 146 147 150 151 152 153 154 155 156 157
0000160 160 161 162 163 164 165 166 167 170 171 172 173 174 175 176 177
0000200
```

```
$ od -c znaki_ASCII
0000000  \0 001 002 003 004 005 006  \a  \b  \t  \n  \v  \f  \r 016 017
0000020 020 021 022 023 024 025 026 027 030 031 032 033 034 035 036 037
0000040       !   "   #   $   %   &   '   (   )   *   +   ,   -   .   /
0000060   0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ?
0000100   @   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
0000120   P   Q   R   S   T   U   V   W   X   Y   Z   [   \   ]   ^   _
0000140   `   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o
0000160   p   q   r   s   t   u   v   w   x   y   z   {   |   }   ~ 177
0000200
```

tail

Polecenie `tail` wyświetla końcową część pliku (domyślnie 10 ostatnich wierszy). Na przykład, aby wyświetlić 40 końcowych wierszy *`pliku`*, należy wykonać następujące polecenie:

```
$ tail -40 plik
```

Przetwarzanie plików

W tym rozdziale omówione są polecenia dotyczące przetwarzania plików.

Polecenie	Znaczenie
`cmp`	Porównanie dwóch plików i wyświetlenie informacji o miejscu pierwszej różnicy między plikami
`csplit`	Dzielenie pliku na fragmenty
`cut`	Wybranie do wyświetlania określonych pól lub znaków z każdego wiersza
`dd`	Kopiowanie, konwersja danych i formatowanie
`diff`	Porównanie dwóch plików i wyświetlenie różnic (dotyczy tylko plików tekstowych)
`file`	Wyświetlenie informacji o typie pliku
`find`	Wyszukiwanie plików określonego typu lub mających nazwę pasującą do wzorca
`fmt`	Formatowanie akapitów w taki sposób, aby wiersze nie przekraczały określonej długości i spełniały kilka innych warunków
`ln`	Tworzenie dowiązania symbolicznego
`nl`	Drukuje numery wierszy plików tekstowych lub tekstu ze standardowego wejścia
`paste`	Łączenie wierszy plików, wyświetlenie ich obok siebie
`sdiff`	Wyświetlenie różnic między plikami
`sort`	Sortowanie wierszy pliku według algorytmu odpowiadającego wybranej opcji
`split`	Dzielenie pliku na zadaną liczbę fragmentów
`strings`	Wyświetlenie wszystkich ciągów znaków drukowalnych, domyślnie o długości co najmniej 4 znaków, zakończonych znakiem nowego wiersza lub znakiem null. Zwykle wykorzystywane do przeglądania plików binarnych w poszukiwaniu łańcuchów ASCII
`tee`	Kopiowanie danych ze standardowego wyjścia do pliku
`touch`	Tworzenie pustego pliku lub uaktualnienie czasu ostatniego dostępu do istniejącego pliku
`tr`	Kopiowanie danych ze standardowego wejścia na standardowe wyjście z jednoczesną zamianą lub usunięciem niektórych znaków
`uniq`	Eliminowanie powtarzających się wierszy z posortowanego pliku
`wc`	Wyświetlenie informacji o liczbie wierszy, słów i znaków w pliku
`which`	Wyświetlenie informacji o tym, który plik zostanie uruchomiony, jeżeli wydane zostanie *`polecenie`*

cmp

Polecenie `cmp` porównuje dwa pliki i informuje o miejscu pojawienia się pierwszej różnicy między nimi. Akceptuje zarówno pliki binarne, jak i tekstowe. Porównywane są kolejno bajty z obu plików. Oto przykład:

```
$ cmp [opcje] plik1 plik2 [pomin1] [pomin2]
```

Opcjonalne argumenty *`pomin1`* i *`pomin2`* określają, ile bajtów należy pominąć w każdym z plików przed rozpoczęciem porównania.

Przykład. W pliku `a` zapisany jest następujący tekst:

```
adam
ola
andrzej
julka
```

Plik `b` ma następującą postać:

```
adam
ola
grzegorz
piotr
```

Porównanie zawartości plików,

```
$ cmp a b
```

daje następujący wynik:

```
a b differ: character 10, line 3
```

csplit

Polecenie `csplit` dzieli plik na fragmenty w miejscach określonych wzorcem, który może być numerem wiersza lub wyrażeniem regularnym. Jeżeli podzielimy plik `a` w miejscu wystąpienia wzorca `andrzej`, wynik ma następującą postać:

```
$ csplit a /andrzej/ {*}
9
14
```

Liczby wyświetlone na ekranie oznaczają liczbę bajtów w plikach powstałych po podziale. Jeśli nie zostanie podany prefiks nazw podzielonych plików po podziale (za pomocą opcji `-f` *`prefiks`*), utworzone pliki otrzymują domyślne nazwy `xx00` i `xx01`. Aby ponownie połączyć je w całość, należy wykonać polecenie

```
$ cat xx00 xx01 > calyplik
```

Oto przykład polecenia, w którym podano prefiks i liczbę wierszy w plikach po podziale:

```
$ csplit -f bb a 2
```

Pliki po podziale otrzymają nazwy `bb00` i `bb01`.

cut

Polecenie `cut` drukuje wybrane kolumny lub pola z jednego lub więcej plików. Na przykład, jeżeli plik `adresy` ma następującą postać:

```
adam adam@host.pl
ola  ola@host.pl
andrzej   andrzej@host.pl
julka     julka@host.pl
```

to polecenie drukowania drugiego pola każdego wiersza daje następujący wynik:

```
$ cut -f2 adresy
adam@host.pl
ola@host.pl
andrzej@host.pl
julka@host.pl
```

Domyślnym separatorem pól jest tabulator `\t`, ale można to zmienić za pomocą opcji `-d`. W tym przykładzie separatorem pól jest spacja:

```
$ cut -f2 -d ' ' adresy
```

W celu wyświetlenia wybranych znaków każdego wiersza, należy użyć opcji `-c`, po czym wpisać numery znaków oddzielone przecinkami:

```
$ cut -c1,2,3,4 adresy
adam
ola
andr
julk
```

Ten sam efekt można osiągnąć za pomocą polecenia

```
$ cut -c1-4 adresy
```

Listę znaków, pól lub bajtów (opcja `-b`) można podać w następującej postaci:

`M`	M-ty bajt, znak lub pole
`M-`	od M-tego bajtu, znaku lub pola do końca wiersza
`M-N`	od M-tego do N-tego bajtu, znaku lub pola
`-N`	od 1 do N-tego bajtu, znaku lub pola

Jeżeli nie ma nazwy pliku lub w to miejsce jest wpisany znak `-`, polecenie `cut` przyjmuje tekst ze standardowego wejścia.

dd

Polecenie `dd` służy do kopiowania i konwersji danych. Dane mogą być pobierane z pliku (opcja `if=`*`plik`*) lub ze standardowego wejścia (domyślnie). Dane wyjściowe kierowane są do pliku (opcja `of=`*`plik`*) lub na standardowe wyjście (domyślnie). Opcje tego polecenia są przedstawione w następującej tabeli:

Opcja	**Znaczenie**	
`bs=`*`n`*	Rozmiar bloków w pliku wejściowym i wyjściowym wynosi *`n`* bajtów	
`cbs=`*`n`*	Bufor konwersji wynosi *`n`* bajtów	
`conv=`*`typkonwersji`*	Określenie typu konwersji; pierwszych pięć znaczników wzajemnie się wyklucza;	
	`ascii`	Konwersja z formatu EBCDIC na format ASCII
	`ebcdic`	Konwersja z formatu ASCII na EBCDIC
	`ibm`	Konwersja z formatu ASCII na EBCDIC w wersji IBM.
	`block`	Konwersja każdego wiersza na wiersze o stałej długości
	`unblock`	Konwersja wierszy o stałej długości na wiersze o zmiennej długości
	`lcase`	Zamiana wielkich liter na małe
	`ucase`	Zamiana małych liter na wielkie
	`swab`	Zamiana bajtów parami
	`noerror`	Przetwarzanie jest kontynuowane także w przypadku pojawienia się błędów
	`notrunc`	Plik wyjściowy nie jest skracany; bloki pliku wyjściowego, które nie zostały zapisane w bieżącym wywołaniu dd, pozostają nienaruszone
	`sync`	Powiększenie rozmiaru bloku wyjściowego do rozmiaru określonego opcją `ibs`
`count=`*`n`*	Kopiowanie *`n`* bloków pliku wejściowego	
`ibs=`*`n`*	Ustawienie rozmiaru bloku wejściowego na *`n`* bajtów (domyślnie 512)	
`if=`*`n`*	Wczytywanie danych wejściowych z *`pliku`*	
`obs=`*`n`*	Ustawienie rozmiaru bloku wyjściowego na *`n`* bajtów (domyślnie 512)	
`of=`*`plik`*	Zapisywanie danych wyjściowych do *`pliku`*	
`seek=`*`n`*	Pominięcie *`n`* bloków pliku wyjściowego	
`skip=`*`n`*	Pominięcie *`n`* bloków pliku wejściowego	

Rozmiary bloków są podawane w bajtach, ze znacznikiem określającym wielokrotność podanej liczby bajtów: w, b lub k oznacza mnożenie przez 2, 512 lub 1024.

Przykłady

Kopiowanie pliku z zamianą wielkich liter na małe:

```
$ dd if=plik1 of=plik2 conv=lcase
```

Kopiowanie obrazu dyskietki instalacyjnej Linuksa na dyskietkę (urządzenie `/dev/fd0`):

```
$ dd if=boot.img of=/dev/fd0 bs=1440k
```

Pominięcie pierwszych 10 bajtów pliku:

```
$ dd ibs=10 skip=1 if=plik1 of=plik2
```

Skopiowanie z *pliku1* 15 bajtów, poczynając od 11-ego bajtu:

```
$ dd if=plik1 of=plik2 bs=1 skip=10 count=15
```

Dopisanie wyniku do pliku2, zachowując jego dotychczasową zawartość:

```
$ dd if=plik1 bs=1 skip=10 count=15 >> plik2
```

Ostatni przykład dotyczy kopiowania dyskietek. Najpierw dane z dyskietki źródłowej są kopiowane do tymczasowego pliku w katalogu `/tmp`:

```
$ dd if=/dev/fd0 of=/tmp/plik
```

Następnie plik jest kopiowany na dyskietkę docelową i usunięty z systemu.

```
$ dd if=/tmp/plik of=/dev/fd0
$ rm /tmp/plik
```

Uwaga: nie należy używać `dd` do kopiowania plików między systemami plików o różnych różnych rozmiarach bloków.

diff

Polecenie `diff` porównuje zawartość dwóch plików, katalogów lub pliku i katalogu. Na przykład, porównanie użytych wcześniej w tym rozdziale plików `a` i `b` daje następujący wynik:

```
$ diff a b
3,4c3,4
< andrzej
< julka
---
> grzegorz
> piotr
```

file

Polecenie `file` wyświetla informację o typie pliku, na przykład:

```
$ file *
a.txt: ASCII text
obrazek.gif: GIF image data, version 89a, 23 x 20,
index.html: HTML document text
zdjecie.jpg: JPEG image data, JFIF standard 1.02, aspect
ratio, 100 x 100
ksiazka.pdf: PDF document, version 1.2
```

W tym przykładzie drukowane są informacje o wszystkich plikach bieżącego katalogu (reprezentowane przez znak `*`). W tym celu wykorzystywane są charakterystyki typów plików (w systemie Red Hat Linux 8.0 zawarte w pliku `/usr/share/magic`).

find

Polecenie `find` służy do wyszukiwania plików. Ma następującą postać:

`find [`*`ścieżka`*`...] [`*`wyrażenie`*`]`

Przeszukiwane jest drzewo katalogowe określone za pomocą *ścieżki*. *Wyrażenie* składa się z opcji określających operację wyszukiwania. W następującym przykładzie wyszukiwany jest plik o nazwie `passwd`. Sprawdzane są wszystkie katalogi systemu.

```
$ find / -name 'passwd' -print
/etc/passwd
```

Oto kilka innych przykładów wyszukiwania.

Pliki rozpoczynające się wielką literą, znajdujące się w katalogach użytkownika `ola`:

```
$ find /home/ola -name '[A-Z]*' -print
```

Pliki w bieżącym katalogu lub jego podkatalogach zawierające *ciąg_znaków*:

```
$ find . -type f -exec grep -l ciąg_znaków {} \
```

Pliki w bieżącym katalogu lub jego podkatalogach zmodyfikowane ponad 10 dni wcześniej:

```
$ find . -type f -ctime +10 –print
```

Pliki, których rozmiar przekracza 1 megabajt:

```
$ find /ścieżka -size 1000000c –print
```

Jeżeli podczas przeszukiwania pojawia się wiele komunikatów typu `permission denied`, co przeszkadza w odczytywaniu wyników, można skierować komunikaty o błędach do urządzenia `/dev/null`:

```
$ find [...] 2>/dev/null
```

fmt

Polecenie `fmt` umożliwia formatowanie tekstu tak, aby wiersz nie przekraczał określonej długości. Oto fragment dokumentacji tego polecenia w systemie Red Hat Linux 8.0:

```
$ info fmt
File: coreutils.info,  Node: fmt invocation,  Next: pr invocation,  Up:
Formatting file contents
```

```
`fmt': Reformat paragraph text
==============================

   `fmt' fills and joins lines to produce output lines of (at most) a
given number of characters (75 by default).  Synopsis:

     fmt [OPTION]... [FILE]...

   `fmt' reads from the specified FILE arguments (or standard input if
none are given), and writes to standard output.

   By default, blank lines, spaces between words, and indentation are
preserved in the output; successive input lines with different
indentation are not joined; tabs are expanded on input and introduced on
output.

   `fmt' prefers breaking lines at the end of a sentence, and tries to
avoid line breaks after the first word of a sentence or before the last
word of a sentence.  A "sentence break" is defined as either the end of
a paragraph or a word ending in any of `.?!', followed by two spaces or
end of line, ignoring any intervening parentheses or quotes.  Like TeX,
`fmt' reads entire "paragraphs" before choosing line breaks; the
algorithm is a variant of that in "Breaking Paragraphs Into Lines"
(Donald E. Knuth and Michael F. Plass, `Software--Practice and
Experience', 11 (1981), 1119-1184).
```

Ten sam tekst można sformatować inaczej. W tym przykładzie długość wiersza jest ograniczona do 40 znaków:

```
$ info fmt | fmt -40
File: coreutils.info,  Node: fmt
invocation,  Next: pr invocation,  Up:
Formatting file contents

`fmt': Reformat paragraph text
==============================

   `fmt' fills and joins lines to
   produce output lines of (at most) a
given number of characters (75 by
default).  Synopsis:

     fmt [OPTION]... [FILE]...

   `fmt' reads from the specified FILE
   arguments (or standard input if
none are given), and writes to standard
output.

   By default, blank lines, spaces
   between words, and indentation are
preserved in the output; successive
input lines with different indentation
are not joined; tabs are expanded on
input and introduced on output.

   `fmt' prefers breaking lines at the
   end of a sentence, and tries to
avoid line breaks after the first word
of a sentence or before the last word
```

```
of a sentence.  A "sentence break"
is defined as either the end of a
paragraph or a word ending in any of
`.?!', followed by two spaces or end
of line, ignoring any intervening
parentheses or quotes.  Like TeX,
`fmt' reads entire "paragraphs"
before choosing line breaks; the
algorithm is a variant of that in
"Breaking Paragraphs Into Lines"
(Donald E. Knuth and Michael F. Plass,
`Software--Practice and Experience',
11 (1981), 1119-1184).
```

ln

Polecenie `ln -s` tworzy dowiązanie symboliczne (jego odpowiednikiem w systemie MS Windows jest skrót) do istniejącego pliku:

```
ln -s plik nowy_plik
```

```
$ ls -l nowy_plik
lrwxrwxrwx   1 ktos  ktos  88 Apr 10 23:41 nowy_plik -> plik
```

paste

Polecenie `paste` wyświetla zawartość odpowiadających sobie wierszy z kilku plików w kolumnach oddzielonych tabulatorem. Na przykład, jeśli `a1` ma następującą postać:

```
ania
ola
ewa
```

a plik `a2` zawiera

```
adam
romek
piotr
```

to wynik polecenia `paste a1 a2` jest następujący:

```
ania    adam
ola     romek
ewa     piotr
```

Użycie opcji `-s` powoduje, że kolejne wiersze pochodzące z jednego pliku umieszczone są w jednym wierszu:

```
$ paste -s a1 a2
ania    ola     ewa
adam    romek   piotr
```

Jeżeli brak jest nazwy pliku lub zamiast nazwy wpisany jest znak −, tekst jest pobierany ze standardowego wejścia. W następującym przykładzie zawartość katalogu jest drukowana w trzech kolumnach:

```
ls | paste - - -
```

Znak | oznacza potok, czyli przekazanie wyniku polecenia do innego polecenia (jest to omówione w rozdziale *Deskryptory plików i przekierowania*).

sdiff

Polecenie `sdiff` *`plik1 plik2`* porównuje dwa pliki i drukuje wynik w następującej postaci:

`tekst  tekst`	Wiersze są identyczne
`tekst <`	Wiersz istnieje tylko w *`pliku1`*
`> tekst`	Wiersz istnieje tylko w *`pliku2`*
`tekst | tekst`	Wiersze są różne

Oto przykład, w którym porównywane są pliki `a` i `b` z wcześniejszej części rozdziału:

```
$ sdiff a b
adam                                                    adam
ola                                                     ola
andrzej                                               | grzegorz
julka                                                 | piotr
```

sort

Polecenie `sort` domyślnie sortuje tekst w kolejności alfabetycznej:

```
$ sort a
adam
andrzej
julka
ola
```

Można jednak wybrać inne warianty uporządkowania tekstu, na przykład w kolejności odwrotnej do alfabetycznej:

```
$ sort -r a
ola
julka
andrzej
adam
```

Podczas sortowania porównywane są najpierw znaki z pierwszej kolumny wszystkich wierszy, następnie znaki z drugiej kolumny itd. Sortowanie wykonywane jest także względem wartości liczbowych. Powiedzmy, że w katalogu znajdują się cztery pliki `a`, `b`, `c` i `d` zawierające 2, 10, 20 i 1000 kilobajtów. Aby je wyświetlić w kolejności od najmniejszego do największego, należy wykonać następujące polecenie:

```
$ ls -s | sort
   2 a
  10 b
  20 c
1000 d
total 1032
```

Polecenie `ls -s` drukuje wielkość pliku i jego nazwę. W ostatnim wierszu pojawia się informacja o łącznej objętości wyświetlanych plików. Aby uporządkować pliki w kolejności odwrotnej, należy użyć opcji `-r`:

```
$ ls -s | sort -r
1000 d
  20 c
  10 b
   2 a
total 1032
```

split

Polecenie `split` służy do dzielenia plików na mniejsze fragmenty. Powiedzmy, że trzeba podzielić plik wykonywalny programu `gimp-1.2`, popularnego narzędzia do edycji grafiki:

```
$ ls –l /usr/bin/gimp-1.2
-rwxr-xr-x    1 root     root      1985708 Aug 31  2002 gimp-1.2
```

Dzielimy na fragmenty:

```
$ split -b 1400000 gimp-1.2
```

Opcja `-b` określa liczbę bajtów w poszczególnych fragmentach. Przed podzieleniem plik zajmował 1985708 bajtów. Powstały dwa fragmenty pliku:

```
-rw-r--r--    1 root     root      1400000 Apr 19 12:02 xaa
-rw-r--r--    1 root     root       585708 Apr 19 12:02 xab
```

Aby te części ponownie złożyć w całość, wystarczy wykonać następujące polecenie:

```
$ cat xaa xab > calyplik
```

strings

Polecenie `strings` drukuje wszyskie sekwencje znaków drukowalnych zawierające co najmniej 4 znaki. Następujące polecenie drukuje ciągi takich znaków w pliku wykonywalnym polecenia `cp`:

```
$ strings /bin/cp
/lib/ld-linux.so.2
libacl.so.1
acl_entries
acl_get_file
_DYNAMIC
acl_set_file
acl_delete_def_file
_init
_fini
acl_from_text
_GLOBAL_OFFSET_TABLE_
acl_free
_Jv_RegisterClasses
acl_extended_file
__gmon_start__
libc.so.6
...
```

Powyższy wydruk zawiera jedynie początkowe wiersze wyniku. Aby wyświetlić tylko sekwencje zawierające co najmniej 20 znaków drukowalnych, należy wykonać następujące polecenie:

```
$ strings -20 /bin/cp
 _GLOBAL_OFFSET_TABLE_
__ctype_get_mb_cur_max
strip-trailing-slashes
setting permissions for %s
cannot make directory %s
missing destination file
missing file arguments
SIMPLE_BACKUP_SUFFIX
abdfHilLprsuvxPRS:V:
Try `%s --help' for more information.
Usage: %s [OPTION]... SOURCE DEST
  or:  %s [OPTION]... SOURCE... DIRECTORY
  or:  %s [OPTION]... --target-directory=DIRECTORY SOURCE...
Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.
Mandatory arguments to long options are mandatory for short options too.
  -a, --archive                same as -dpR
      --backup[=CONTROL]       make a backup of each existing destination
file
  -b                           like --backup but does not accept an argu-
ment
      --copy-contents          copy contents of special files when recur-
sive
  -d                           same as --no-dereference --preserve=link
      --no-dereference         never follow symbolic links
  -f, --force                  if an existing destination file cannot be
                                 opened, remove it and try again
  -i, --interactive            prompt before overwrite
```

```
  -H                          follow command-line symbolic links
  -l, --link                  link files instead of copying
  -L, --dereference           always follow symbolic links
  -p                          same as --preserve=mode,ownership,timestamps
      --preserve[=ATTR_LIST]  preserve the specified attributes (default:
                                mode,ownership,timestamps), if possible
                                additional attributes: links, all
      --no-preserve=ATTR_LIST don't preserve the specified attributes
      --parents               append source path to DIRECTORY
  -P                          same as `--no-dereference'
  -R, -r, --recursive         copy directories recursively
      --remove-destination    remove each existing destination file before
                                attempting to open it (contrast with --
force)
      --reply={yes,no,query}  specify how to handle the prompt about an
                                existing destination file
      --sparse=WHEN           control creation of sparse files
      --strip-trailing-slashes remove any trailing slashes from each SOURCE
                                argument
  -s, --symbolic-link         make symbolic links instead of copying
  -S, --suffix=SUFFIX         override the usual backup suffix
      --target-directory=DIRECTORY  move all SOURCE arguments into DIRECTO-
RY
...
```

Wynik został przytoczony w niepełnej postaci.

touch

Polecenie `touch` zmienia czas modyfikacji lub czas ostatniego dostępu do pliku, nie zmieniając przy tym zawartości pliku. Oto przykład zmiany daty i czasu ostatniej modyfikacji pliku:

```
$ ls -l plik1
-rw-------    1 lsb   Brak           23 Feb 21 19:38 plik1

$ touch plik1

$ ls -l
-rw-------    1 lsb   Brak           24 Feb 21 20:01 plik1
```

Jeżeli plik nie istnieje, polecenie `touch` *`plik`* powoduje utworzenie pustego pliku o tej nazwie.

tr

Polecenie `tr` zamienia lub usuwa znaki z wejściowego strumienia danych. Wynik można zapisać do pliku. Oto kilka przykładów.

Zamiana wielkich liter na małe. Wynik polecenia `cat` jest przekazany na wejście polecenia `tr` (zobacz rozdział *Deskryptory plików i przekierowania*)

```
$ cat plik | tr '[A-Z]' '[a-z]'
```

Zamiana małych liter na wielkie (*plik1* jest plikiem wejściowym, *plik2* – plikiem wyjściowym):

```
$ cat plik1 | tr '[a-z]' '[A-Z]' > plik2
```

Zamiana spacji na znaki nowego wiersza (kod ASCII 012). W wyniku każdy ciąg znaków jest umieszczony w osobnym wierszu:

```
$ tr ' ' '\012' < plik1 > plik2
```

Usunięcie dwukropków:

```
$ tr -d : < plik1 > plik2
```

Usunięcie spacji:

```
$ tr -d ' ' < plik1 > plik2
```

uniq

Polecenie `uniq` usuwa powtarzające się wiersze pliku. Na przykład, jeżeli plik `imiona` ma następującą postać:

```
ola
ola
andrzej
piotr
grzegorz
```

to wynikiem polecenia `uniq imiona` jest następujący tekst:

```
ola
andrzej
piotr
grzegorz
```

Oto inny przykład. Drukujemy listę zalogowanych użytkowników:

```
$ who
operator  pts/5     May 18 16:09   (eye-en0.man.poznan.pl)
piotr     pts/6     May 17 11:02   (hawthorn.man.poznan.pl)
julka     pts/11    May 14 21:30   (almond.man.poznan.pl)
jerzy     pts/13    May 19 10:05   (c9-108.icpnet.pl)
krzysztof pts/14    May 19 09:17   (almond.man.poznan.pl)
jerzy     pts/17    May 19 14:59   (c9-108.icpnet.pl)
```

Aby wydrukować tylko nazwy użytkowników, można wykonać następujące polecenie:

```
$ who | awk '{ print $1 }'
operator
```

```
piotr
julka
jerzy
krzysztof
jerzy
```

Polecenie `awk '{ print $1 }'` drukuje pierwsze pole każdego wiersza (zobacz rozdział *Język Awk*). Użytkownik `jerzy` jest zalogowany dwukrotnie. Co należy zrobić, aby lista zawierała po jednej nazwie każdego z zalogowanych użytkowników? Uporządkować nazwy w kolejności alfabetycznej za pomocą polecenia `sort` i wykonać polecenie `uniq`:

```
$ who | awk '{ print $1 }' | sort | uniq
jerzy
julka
krzysztof
operator
piotr
```

Przekazywanie wyników jednego polecenia na wejście innego polecenia jest omówione w rozdziale *Deskryptory plików i przekierowania*.

wc

Polecenie `wc` wyświetla liczbę znaków, słów i wierszy w pliku:

```
$ wc a.txt
213  1009 7757 a.txt
```

W tym przykładzie `a.txt` zawiera 213 wierszy, 1009 słów, 7757 znaków. Strumień danych może być również przekazany ze standardowego wejścia:

```
$ cat a.txt | wc
213  1009 7757 a.txt
```

Polecenie `wc -l` oblicza liczbę wierszy tekstu. Na przykład, aby wyświetlić liczbę zalogowanych użytkowników, można dołączyć `wc -l` do ostatniego przykładu dotyczącego polecenia `uniq`:

```
$ who | awk '{ print $1 }' | sort | uniq | wc -l
5
```

which

Za pomocą polecenia `which` można sprawdzić, który plik zostanie uruchomiony, jeżeli zostanie wydane *`polecenie`*, na przykład:

```
$ which ls
/usr/bin/ls
```

Odpowiedź oznacza, że polecenie `ls` jest obsługiwane przez plik wykonywalny `ls`, znajdujący się w katalogu `/usr/bin`.

Drukowanie

Polecenie	Znaczenie
`lpq` lub `lpstat`	Wyświetlenie stanu kolejki drukowania
`lpr` lub `lp`	Wysłanie pliku do drukarki
`lprm` lub `cancel`	Usunięcie zadania drukowania z kolejki
`pr`	Formatowanie tekstu i drukowanie na standardowym wyjściu

Oto ogólna postać każdego z poleceń związanych z drukowaniem:

```
lpq [opcje]
```

```
lpr [opcje] [pliki]
```

```
cancel [nr_zadania] [drukarka]
```

Informacje o numerach zadań i drukarkach można uzyskać za pomocą polecenia `lpstat`.

```
pr [opcje] [plik]
```

Polecenia dotyczące stanu systemu i jego zasobów

Polecenie	Znaczenie
`chsh`	Zmiana domyślnej powłoki użytkownika; administrator może odebrać użytkownikom prawo wykonywania tego polecenia
`date`	Wyświetlenie bieżącego dnia tygodnia, godziny i daty lub ustawienie bieżącego czasu i daty w systemie (tylko administrator może to zrobić)
`df`	Wyświetlenie informacji o zamontowanych systemach plików
`du`	Wyświetlenie informacji o rozmiarze miejsca na dysku zajętego przez katalog bieżący i jego podkatalogi
`hostname`	Wyświetlenie nazwy hosta
`kill`	Zamknięcie czynnego procesu
`man` *polecenie*	Wyświetlenie opisu składni *polecenia*
`nice` *polecenie*	Wykonanie *polecenia* lub programu na niższym priorytecie
`nohup` *polecenie*	Uruchomione *polecenie* jest kontynuowane nawet po wylogowaniu użytkownika
`passwd`	Zmiana hasła użytkownika
`ps`	Wyświetlenie informacji o uruchomionych procesach
`renice`	Zmiana priorytetu działającego procesu
`stty`	Zmiana parametrów terminala lub wyświetlenie bieżących definicji
`top`	Wyświetlenie informacji o procesach najbardziej obciążających CPU
`uname`	Wyświetlenie nazwy komputera i innych informacji o sprzęcie i oprogramowaniu systemowym
`uptime`	Wyświetlenie czasu, jaki upłynął od ostatniego uruchomienia systemu
`w`	Wyświetlenie informacji o obciążeniu systemu, zalogowanych użytkownikach i uruchomionych przez nich programach
`which` *polecenie*	Wyświetlenie nazwy pliku uruchamianego po wywołaniu *polecenia*
`who`	Wyświetlenie listy zalogowanych użytkowników
`whoami`	Wyświetlenie nazwy bieżącego użytkownika

date

Polecenie `date` wyświetla bieżącą datę i czas:

```
$ date
Sat Feb 22 12:11:37 2003
```

Polecenie `date` ma parędziesiąt opcji, dzięki czemu można szczegółowo określić format wyświetlania daty i czasu. Oto przykłady:

```
$ date +%H:%M:%S
11:28:53
```

Wyświetla bieżący czas w formacie godzina:minuta:sekunda

```
$ date +%d/%m/%Y"   "%H:%M:%S
25/04/2003   13:31:03
```

Wyświetla bieżącą datę w formacie dzień/miesiąc/rok i czas w formacie godzina:minuta:sekunda

df

Polecenie `df` drukuje informację o stopniu wykorzystania bloków i i-węzłów w każdym zamontowanym systemie plików. Szczegóły formatu, w jakim drukowany jest wynik, zależą od konkretnego systemu i wersji programu. W następującym przykładzie system został podzielony na dwie partycje `/dev/hda2` i `/dev/hda3`, zamontowane w punktach `/` i `/usr`.

```
$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/hda2              3028108   1388456   1485832  49% /
/dev/hda3              4063872   1825104   2032336  48% /usr
```

du

Polecenie `du` wyświetla informację o tym, ile miejsca na dysku (zwykle w kilobajtach) zajmują wskazane pliki lub katalogi. Wykonanie tego polecenia bez opcji i argumentów powoduje wydrukowanie informacji o wielkości poszczególnych katalogów w całym drzewie katalogowym należącym do bieżącego katalogu. Aby wyświetlić wielkość każdego z plików, należy użyć opcji `-a`. Aby wydrukować jedynie sumę zajętego miejsca na dysku, użyj opcji `-s`, na przykład:

```
$ du -s /etc
11916
```

Katalog `/etc` i wszystkie jego podkatalogi zajmują łącznie prawie 12 megabajtów.

kill

Polecenie `kill` przesyła sygnał zamknięcia do działającego procesu. Na przykład,

```
$ kill -9 938
```

wysyła do procesu o identyfikatorze 938 sygnał bezwarunkowego zamknięcia procesu. Aby uzyskać pełną listę możliwych sygnałów, należy wykonać polecenie `kill -l`. Źródłem informacji o tych sygnałach jest plik `/usr/include/sys/signal.h`. Oto wynik uzyskany w systemie Red Hat Linux 8.0:

```
$ kill -l
 1) SIGHUP  2) SIGINT  3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS  8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG 24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR 31) SIGSYS 32) SIGRTMIN    33) SIGRTMIN+1
34) SIGRTMIN+2  35) SIGRTMIN+3  36) SIGRTMIN+4  37) SIGRTMIN+5
38) SIGRTMIN+6  39) SIGRTMIN+7  40) SIGRTMIN+8  41) SIGRTMIN+9
42) SIGRTMIN+10 43) SIGRTMIN+11 44) SIGRTMIN+12 45) SIGRTMIN+13
46) SIGRTMIN+14 47) SIGRTMIN+15 48) SIGRTMAX-15 49) SIGRTMAX-14
50) SIGRTMAX-13 51) SIGRTMAX-12 52) SIGRTMAX-11 53) SIGRTMAX-10
54) SIGRTMAX-9  55) SIGRTMAX-8  56) SIGRTMAX-7  57) SIGRTMAX-6
```

```
58) SIGRTMAX-5  59) SIGRTMAX-4  60) SIGRTMAX-3  61) SIGRTMAX-2
62) SIGRTMAX-1  63) SIGRTMAX
```

Opis wszystkich opcji znajduje się w pomocy podręcznej (polecenie `man kill`).

nice

Polecenie `nice` obniża priorytet procesu. Opcja `-`*`n`* ustawia priorytet procesu. Zwykle liczba *n* ma wartości z przedziału od 1 do 19 (wówczas domyślnym priorytetem jest 10) lub od –20 do 19 (wówczas domyślnym priorytetem jest 0). Zwykły użytkownik może jedynie obniżyć priorytet. Im większe *n*, tym niższy priorytet. W przypadku pominięcia tej opcji priorytet jest obniżany o wartość domyślną (zwykle o 10). Administrator może podwyższyć priorytet procesu, podając ujemne wartości *n*, na przykład

```
$ nice -10 polecenie
```

nohup

Polecenie `nohup` jest przydatne w sytuacji, w której uruchomiony proces ma być kontynuowany nawet po wylogowaniu użytkownika. Standardowe wyjście należałoby wówczas skierować do pliku:

```
$ nohup polecenie [argumenty] > plik &
```

Znak & (ampersand) oznacza uruchomienie zadania w tle.

ps

Polecenie `ps` drukuje informację o stanie poszczególnych procesów. Opcja `-e` powoduje wypisanie skrótowych informacji o wszystkich procesach, a opcja `-f` daje pełną informację o każdym z procesów. Oto przykłady wykonania tych poleceń (w systemie Red Hat 8.0):

```
$ ps
  PID TTY          TIME CMD
 1165 pts/1    00:00:00 bash
 1560 pts/1    00:00:00 ps

$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
root      1165  1136  0 11:32 pts/1    00:00:00 bash
root      1564  1165  0 12:37 pts/1    00:00:00 ps _f

$ ps -e
  PID TTY          TIME CMD
    1 ?        00:00:05 init
    2 ?        00:00:00 keventd
    3 ?        00:00:00 kapmd
    4 ?        00:00:00 ksoftirqd_CPU0
    5 ?        00:00:01 kswapd
    6 ?        00:00:00 bdflush
```

```
    7 ?        00:00:00 kupdated
    8 ?        00:00:00 mdrecoveryd
  340 ?        00:00:00 syslogd
  344 ?        00:00:00 klogd
  361 ?        00:00:00 portmap
  381 ?        00:00:00 rpc.statd
  436 ?        00:00:00 cardmgr
  474 ?        00:00:00 apmd
  525 ?        00:00:00 sshd
  539 ?        00:00:00 xinetd
  562 ?        00:00:00 sendmail
  572 ?        00:00:00 sendmail
  582 ?        00:00:00 gpm
  591 ?        00:00:00 crond
  636 ?        00:00:17 xfs
  654 ?        00:00:00 atd
  663 ?        00:00:00 login
  664 tty2     00:00:00 mingetty
  665 tty3     00:00:00 mingetty
  666 tty4     00:00:00 mingetty
  667 tty5     00:00:00 mingetty
  668 tty6     00:00:00 mingetty
  673 tty1     00:00:00 bash
  979 tty1     00:00:00 startx
  990 tty1     00:00:00 xinit
  991 ?        00:00:18 X
  994 tty1     00:00:02 gnome-session
 1010 ?        00:00:00 ssh-agent
 1021 tty1     00:00:02 gconfd-2
 1023 ?        00:00:00 bonobo-activati
 1025 ?        00:00:02 metacity
 1027 ?        00:00:04 gnome-settings-
 1031 ?        00:00:00 fam
 1121 ?        00:00:04 gnome-panel
 1123 ?        00:00:10 nautilus
 1125 ?        00:00:01 magicdev
 1127 ?        00:00:00 pam-panel-icon
 1129 ?        00:00:05 rhn-applet-gui
 1130 ?        00:00:00 pam_timestamp_c
 1136 ?        00:00:07 gnome-terminal
 1137 pts/0    00:00:00 bash
 1165 pts/1    00:00:00 bash
 1542 pts/0    00:00:00 man
 1545 pts/0    00:00:00 sh
 1546 pts/0    00:00:00 sh
 1550 pts/0    00:00:00 less
 1563 pts/1    00:00:00 ps

$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 11:00 ?        00:00:05 init
```

```
root         2     1  0 11:00 ?        00:00:00 [keventd]
root         3     1  0 11:00 ?        00:00:00 [kapmd]
root         4     1  0 11:00 ?        00:00:00 [ksoftirqd_CPU0]
root         5     1  0 11:00 ?        00:00:01 [kswapd]
root         6     1  0 11:00 ?        00:00:00 [bdflush]
root         7     1  0 11:00 ?        00:00:00 [kupdated]
root         8     1  0 11:00 ?        00:00:00 [mdrecoveryd]
root       340     1  0 11:01 ?        00:00:00 syslogd -m 0
root       344     1  0 11:01 ?        00:00:00 klogd -x
rpc        361     1  0 11:01 ?        00:00:00 portmap
rpcuser    381     1  0 11:01 ?        00:00:00 rpc.statd
root       436     1  0 11:01 ?        00:00:00 /sbin/cardmgr
root       474     1  0 11:01 ?        00:00:00 /usr/sbin/apmd -p
root       525     1  0 11:01 ?        00:00:00 /usr/sbin/sshd
root       539     1  0 11:01 ?        00:00:00 xinetd -stayalive
root       562     1  0 11:01 ?        00:00:00 sendmail: accepti
smmsp      572     1  0 11:01 ?        00:00:00 sendmail: Queue r
root       582     1  0 11:01 ?        00:00:00 gpm -t ps/2 -m /d
root       591     1  0 11:01 ?        00:00:00 crond
xfs        636     1  0 11:01 ?        00:00:17 xfs -droppriv -da
daemon     654     1  0 11:01 ?        00:00:00 /usr/sbin/atd
root       663     1  0 11:01 ?        00:00:00 login -- root
root       664     1  0 11:01 tty2     00:00:00 /sbin/mingetty tt
root       665     1  0 11:01 tty3     00:00:00 /sbin/mingetty tt
root       666     1  0 11:01 tty4     00:00:00 /sbin/mingetty tt
root       667     1  0 11:01 tty5     00:00:00 /sbin/mingetty tt
root       668     1  0 11:01 tty6     00:00:00 /sbin/mingetty tt
root       673   663  0 11:02 tty1     00:00:00 -bash
root       979   673  0 11:30 tty1     00:00:00 /bin/sh /usr/X11R
root       990   979  0 11:30 tty1     00:00:00 xinit /etc/X11/xi
root       991   990  0 11:30 ?        00:00:18 X :0
root       994   990  0 11:31 tty1     00:00:02 /usr/bin/gnome-se
root      1010   994  0 11:31 ?        00:00:00 ssh-agent /etc/X1
root      1021     1  0 11:31 tty1     00:00:02 /usr/libexec/gcon
root      1023     1  0 11:31 ?        00:00:00 /usr/libexec/bono
root      1025     1  0 11:31 ?        00:00:02 /usr/bin/metacity
root      1027     1  0 11:31 ?        00:00:04 gnome-settings-da
root      1031   539  0 11:31 ?        00:00:00 fam
root      1121     1  0 11:31 ?        00:00:04 gnome-panel --sm-
root      1123     1  0 11:31 ?        00:00:10 nautilus --no-def
root      1125     1  0 11:31 ?        00:00:01 magicdev --sm-cli
root      1127     1  0 11:31 ?        00:00:00 pam-panel-icon --
root      1129     1  0 11:31 ?        00:00:04 /usr/bin/python /
root      1130  1127  0 11:31 ?        00:00:00 /sbin/pam_timesta
root      1136     1  0 11:32 ?        00:00:07 /usr/bin/gnome-te
root      1137  1136  0 11:32 pts/0    00:00:00 bash
root      1165  1136  0 11:32 pts/1    00:00:00 bash
root      1542  1137  0 12:31 pts/0    00:00:00 man date
root      1545  1542  0 12:31 pts/0    00:00:00 sh -c (cd /usr/sh
root      1546  1545  0 12:31 pts/0    00:00:00 sh -c (cd /usr/sh
root      1550  1546  0 12:31 pts/0    00:00:00 /usr/bin/less -is
root      1562  1165  0 12:36 pts/1    00:00:00 ps -ef
```

Opis polecenia `ps` jest można wyświetlić za pomocą poleceń `man ps` lub `ps --help`.

`renice`

Polecenie `renice` zmienia priorytet działającego procesu. Można zmienić priorytet jednego procesu, całej ich grupy lub wszystkich procesów danego użytkownika. Na przykład,

```
$ renice +5 -p 938
```

Obniża o 5 priorytet procesu nr 938. Następujące polecenie:

```
$ renice +10 -u ola
```

Obniża o 10 priorytety wszystkich procesów użytkownika `ola`.

Edytor tekstu vi

Edytor `vi` działa w dwóch trybach: wpisywania tekstu i wpisywania poleceń. Po otwarciu pliku edytor funkcjonuje w trybie poleceń. Aby przejść do trybu wpisywania tekstu, należy nacisnąć jeden z klawiszy **i**, **I**, **a**, **A**, **o**, **O** (zobacz podrozdział *Polecenia edycji*). Aby przejść z trybu wpisywania tekstu do trybu poleceń, należy nacisnąć klawisz **Esc**. Z trybu wpisywania można wyjść także za pomocą kombinacji klawiszy **Ctrl+C**.

Edytor `vi` służy do edycji plików zapisanych w formacie ASCII. Nie można nim dokonywać edycji plików binarnych. Oto niektóre ograniczenia tego edytora:

Maksymalna długość wiersza	Około 1020 znaków
Maksymalny rozmiar pliku	Zależnie od systemu
Długość nazwy pliku	Zwykle 128 znaków
Bufor insert/delete	128 znaków
Definiowanie makr	Zwykle do 32 makr za pomocą polecenia `map`
Łączna długość makr	512 znaków

Aby vi funkcjonował poprawnie, zmienna środowiskowa `TERM` powinna być ustawiona na `vt100`. Możliwe są inne ustawienia, ale ten typ terminala jest najbardziej rozpowszechniony i sprawia najmniej kłopotów.

Edytor uruchamiany jest następującym poleceniem:

`vi [`*`opcje`*`] `*`plik`*

Aby zabezpieczyć się przed przypadkowym zapisaniem pliku, można wywołać edytor poleceniem

`view `*`plik`*

Jest to odpowiednik polecenia `vi -R`. Podczas zapisywania pliku należy podać nazwę pliku lub potwierdzić chęć zapisania pliku pod poprzednią nazwą. W przypadku pracy z plikami sporych rozmiarów warto znać polecenie

`vi +`*`n plik`*

gdzie *`n`* jest numerem wiersza. Jeżeli po otwarciu pliku kursor ma się znaleźć w wierszu numer 500, należy wykonać polecenie

`vi +500 `*`plik`*

Następujące polecenie ustawia kursor w miejscu pierwszego pojawienia się łańcucha *`wzorzec`*:

`vi +/`*`wzorzec plik`*

Jeżeli zdarzy się nieszczęśliwy wypadek przy pracy, np. w wyniku awarii systemu i konieczne jest odzyskiwanie zmian dokumentu, należy wykonać następujące polecenie

```
vi -r plik
```

Zmiana położenia kursora

W następującej tabeli zawarte są polecenia zmiany położenia kursora:

Polecenie	Znaczenie
n	Przesunięcie kursora *n* wierszy do przodu, gdzie *n* jest liczbą. Na przykład, wpisanie liczby 300 i naciśnięcie klawisza **Enter** powoduje skok 300 wierszy do przodu
n-	Przesunięcie kursora *n* wierszy wstecz
h	Przesunięcie kursora jeden znak w lewo
j	Przesunięcie kursora jeden wiersz do dołu
k	Przesunięcie kursora jeden wiersz do góry
l	Przesunięcie kursora jeden znak w prawo
Backspace	Przesunięcie kursora jeden znak wstecz
spacja	Przesunięcie kursora jeden znak do przodu
0	Przesunięcie kursora na początek wiersza
^	Przesunięcie kursora na początek wiersza
$	Przesunięcie kursora na koniec wiersza
w	Przesunięcie kursora jedno słowo do przodu; słowo jest tu rozumiane jako nieprzerwany ciąg znaków, od odstępu do odstępu
b	Przesunięcie kursora na początek poprzedniego słowa
e	Przesunięcie kursora na koniec słowa
+	Przesunięcie kursora na początek następnego wiersza
-	Przesunięcie kursora na początek poprzedniego wiersza
(	Przesunięcie kursora na początek zdania; początek zdania jest rozumiany jako pierwszy znak po kropce różny od odstępu
)	Przesunięcie kursora na początek następnego zdania
{	Przesunięcie kursora na początek akapitu
}	Przesunięcie kursora na początek następnego akapitu
n|	Przesunięcie kursora do *n*-tej kolumny bieżącego wiersza
Ctrl+F	Przesunięcie kursora jeden ekran do przodu
Ctrl+B	Przesunięcie kursora jeden ekran wstecz
Ctrl+D	Przesunięcie kursora pół ekranu do przodu
Ctrl+U	Przesunięcie kursora pół ekranu wstecz
Ctrl+E	Przewinięcie tekstu jeden wiersz do przodu
Ctrl+Y	Przewinięcie tekstu jeden wiersz wstecz
z+Enter	Przesunięcie wiersza, w którym znajduje się kursor, do góry ekranu
z.	Przesunięcie wiersza, w którym znajduje się kursor, na środek ekranu
z-	Przesunięcie wiersza, w którym znajduje się kursor, do dołu ekranu
H	Umieszczenie kursora w lewym górnym rogu ekranu
M	Umieszczenie kursora w środkowym wierszu ekranu
L	Umieszczenie kursora w lewym dolnym rogu ekranu
G	Umieszczenie kursora na początku ostatniego wiersza pliku
*n*G	Umieszczenie kursora na początku *n*-tego wiersza
Ctrl+G	Wyświetlenie nazwy edytowanego pliku, numeru bieżącego wiersza, liczby wszystkich wierszy w pliku i numeru kolumny, w której znajduje się kursor
Ctrl+L	Ponowne rysowanie ekranu

Wyszukiwanie tekstu w pliku

`/`*`tekst`*	Wyszukiwanie ciągu *`tekst`* do przodu
`n`	Powtórzenie ostatniego wyszukiwania
`N`	Powtórzenie ostatniego wyszukiwania w przeciwnym kierunku
`/`	Powtórzenie wyszukiwania do przodu
`?`*`tekst`*	Wyszukiwanie ciągu *`tekst`* wstecz
`?`	Powtórzenie wyszukiwania wstecz
`%`	Znalezienie pary dla nawiasu zwykłego, klamrowego lub kwadratowego

Polecenia edycji

Następujące polecenia (wykonywane w trybie poleceń) powodują przejście do trybu wpisywania:

Polecenie	**Znaczenie**
`i`	Rozpoczęcie pisania tekstu przed kursorem
`a`	Rozpoczęcie pisania tekstu za kursorem
`I`	Rozpoczęcie pisania tekstu na początku wiersza
`A`	Rozpoczęcie pisania tekstu na końcu wiersza
`o`	Otwarcie nowego wiersza poniżej kursora
`O`	Otwarcie nowego wiersza powyżej kursora
`R`	Rozpoczęcie wpisywania tekstu w bieżącym położeniu kursora z usunięciem dotychczasowego tekstu do końca wiersza

Spośród możliwości dostępnych w trybie wpisywania należy pamiętać o następujących opcjach:

`Esc`	Wyjście z trybu wpisywania
`Enter`	Otwarcie nowego wiersza

Zmiana i usuwanie tekstu

Polecenie	**Znaczenie**
`cw`	Usunięcie od kursora do końca słowa i wejście w tryb wpisywania
`cc`	Usunięcie bieżącego wiersza i wejście w tryb wpisywania
`C`	Usunięcie części wiersza, od kursora do końca wiersza i wejście w tryb wpisywania
`dw`	Usunięcie od kursora do końca słowa
`dd`	Usunięcie bieżącego wiersza
`n``dd`	Usunięcie *`n`* wierszy, na przykład `20dd` oznacza usunięcie 20 wierszy
`dk`	Usunięcie wiersza powyżej kursora
`D`	Usunięcie części wiersza od kursora do końca wiersza
`d^`	Usunięcie od początku wiersza do kursora
`d$`	Usunięcie od kursora do końca wiersza; polecenie równoważne `D`

`d+`	Usunięcie wiersza bieżącego i następnego
`d-`	Usunięcie wiersza poprzedniego i bieżącego
`d/`*`tekst`*	Usunięcie wszystkich znaków od kursora do pierwszego wystąpienia ciągu *`tekst`*
`dG`	Usunięcie wierszy, poczynając od bieżącego wiersza do końca pliku
`d1G`	Usunięcie wierszy, od początku pliku do bieżącego wiersza
`d`*`n`*`G`	Usunięcie wierszy między wierszem *`n`*-tym a bieżącym, niezależnie od tego, czy *`n`*-ty wiersz znajduje się przed kursorem, czy za nim, na przykład `d10G`
`x`	Usunięcie pojedynczego znaku, znajdującego się w bieżącym położeniu kursora
`X`	Usunięcie jednego znaku wstecz
`p`	Wstawienie ostatnio usuniętego tekstu za kursorem
`P`	Wstawienie ostatnio usuniętego tekstu przed kursorem
`s`	Wstawienie tekstu w miejsce bieżącego znaku
`S`	Wstawienie znaku w miejsce bieżącego wiersza
`u`	Anulowanie ostatniej zmiany
`U`	Anulowanie wszystkich zmian dokonanych w bieżącym wierszu
`~`	Zmiana wielkości bieżącego znaku
`&`	Powtórzenie ostatniego polecenia `:s`
`:e!`	Anulowanie wszystkich zmian dokonanych od momentu ostatniego zapisu pliku
`:s/`*`tekst1`*`/`*`tekst2`*`/`	Zastąpienie ciągu *`tekst1`* ciągiem *`tekst2`*
`:s/`*`tekst1`*`/`*`tekst2`*`/g`	Zastąpienie ciągu *`tekst1`* ciągiem *`tekst2`* w całym bieżącym wierszu
`:g/`*`tekst1`*`/ s//`*`tekst2`*	Zastąpienie ciągu *`tekst1`* ciągiem *`tekst2`* w całym pliku; w każdym wierszu podstawienie jest wykonywane tylko raz, dla pierwszego wystąpienia ciągu *`tekst1`*
`:g/`*`tekst1`*`/ s//`*`tekst2`*`/g`	Zastąpienie ciągu *`tekst1`* ciągiem *`tekst2`* w całym pliku
`:.,.+`*`n`* `s/`*`tekst1`*`/`*`tekst2`*`/g`	Zastąpienie wszystkich wystąpień ciągu *`tekst1`* ciągiem *`tekst2`* w kolejnych *`n`*+1 wierszach, poczynając od bieżącego wiersza

Kopiowanie i przenoszenie tekstu

Polecenie	**Znaczenie**
`:r` *`plik`*	Wczytanie pliku w bieżącym położeniu kursora
`Y`	Skopiowanie bieżącego wiersza do bufora
`n``Y`	Skopiowanie *`n`* kolejnych wierszy do bufora
`yy`	Skopiowanie bieżącego wiersza
`"ay`	Skopiowanie wiersza do bufora `a`
`"b`*`n`*`yy`	Skopiowanie `n` kolejnych wierszy do bufora `b`
`"Ay`	Dołączenie wiersza do bufora `a`
`"ap`	Wstawienie przed kursorem tekstu z bufora `a`
`"aP`	Wstawienie za kursorem tekstu z bufora `a`

Dostępnych jest 36 buforów do kopiowania i przenoszenia tekstu: jeden domyślny, bez nazwy i 35 o nazwach od `a` do `z` i od `1` do `9`. Użycie w nazwie bufora wielkiej litery, np. `A`, powoduje dołączenie tekstu do zawartości bufora (w tym przykładzie do bufora `a`). Jeśli przed poleceniem usuwającym tekst umieszczony jest znak `"` i nazwa bufora, np. `"add`, to usunięty tekst trafia do bufora i może być następnie skopiowany z bufora w dowolnym miejscu pliku.

Przykłady

`"a8dd`	Usunięcie 8 wierszy i umieszczenie ich w buforze `a`
`"kdG`	Usunięcie wierszy, od bieżącego do końca pliku i umieszczenie ich w buforze `k`

Zapisywanie plików

Polecenie	**Znaczenie**
`ZZ`	Zapis do pliku i zamknięcie edytora
`:x`	To samo, co `ZZ`
`:wq`	Zapis do pliku i wyjście z edytora
`:w`	Zapis bieżącego stanu dokumentu
`:w` *`plik`*	Zapis bieżącego stanu dokumentu do *`pliku`*
`:`*`n1`*`,`*`n2`*`w` *`plik`*	Zapis wierszy od *`n1`* do *`n2`* do *`pliku`*
`:`*`n1`*`,`*`n2`*`w >>` *`plik`*	Dopisanie wierszy od *`n1`* do *`n2`* do *`pliku`*
`:w!`	Wymuszenie zapisu bez względu na prawa dostępu
`:q`	Wyjście z edytora, pod warunkiem, że zmiany dokonane w pliku zostały już zapisane
`:q!`	Wyjście z edytora z anulowaniem dokonanych zmian
`:r` *`plik`*	Wczytanie treści *`pliku`* poniżej kursora
`:f`	Wyświetlenie nazwy bieżącego pliku
`:f` *`plik`*	Zmiana nazwy bieżącego pliku na *`plik`*

Jednoczesna edycja wielu plików

Możliwa jest jednoczesna praca z wieloma plikami. Należy je wywołać podczas uruchamiania edytora:

```
$ vi plik1 plik2 plik3
```

Przejście do edycji kolejnego pliku odbywa się po wpisaniu polecenia `:e nazwa_pliku`. Można też początkowo wywołać tylko jeden plik, a następnie przejść do edycji kolejnego za pomocą polecenia `:e`.

`:e` *`plik`*	Przejście do edycji innego *`pliku`*
`:n`	Edycja następnego pliku
`:args`	Wyświetlenie nazw wszystkich plików z bieżącej sesji edytora

Inne polecenia

Polecenie	Znaczenie
`J`	Połączenie dwóch wierszy, bieżącego i następnego
`>>`	Przesunięcie bieżącego wiersza w prawo o szerokość tabulatora
`<<`	Przesunięcie bieżącego wiersza w prawo o szerokość tabulatora
`>}`	Przesunięcie wszystkich wierszy w prawo o szerokość tabulatora, od bieżącego wiersza do końca akapitu
`:!`*`polecenie`*	Wykonanie polecenia powłoki i powrót do edycji pliku
`:r!`*`polecenie`*	Wykonanie polecenia powłoki i wstawienie wyniku w miejscu położeniu kursora
`:set number`	Wyświetlenie numerów wierszy
`:set nonumber`	Anulowanie wyświetlania numerów wierszy
`:sh`	Przejście do powłoki Bourne'a; aby wyjść z powłoki i powrócić do edycji, należy wykonać polecenie `exit` lub nacisnąć kombinację klawiszy **Ctrl+D**
`:ab pt Pan Tadeusz`	Definiowanie skrótu; po wpisaniu liter `pt` i naciśnięciu klawisza **Enter**, tabulatora lub spacji wpisane zostanie wyrażenie `Pan Tadeusz`
`:unab pt`	Usunięcie definicji skrótu `pt`

Plik konfiguracji `.exrc`

Jeżeli w katalogu głównym użytkownika istnieje plik `.exrc`, zawarte w nim polecenia są wykonywane przy uruchomieniu edytora vi. W tabeli umieszczone są przykłady poleceń, jakie można umieścić w pliku `.exrc`.

Polecenie	Znaczenie
`:set number` lub `:set nu`	Włącza numerowanie wierszy
`:set nonumber` lub `:set nonu`	Wyłącza numerowanie wierszy
`:syntax on`	Włącza kolorowanie elementów składni w językach programowania
`:syntax off`	Wyłącza kolorowanie składni
`:set autoindent` lub `:set ai`	Włącza automatyczne wcięcia
`:set noautoindent` lub `:set noai`	Wyłącza automatyczne wcięcia
`:set wrapscan` lub `:set ws`	Po włączeniu tej opcji wyszukiwanie wzorców za pomocą poleceń `/`, `?`, `n` lub `N` jest kontynuowane po dojściu do końca (lub początku) pliku
`:set nowrapscan` lub `:set nows`	Wyłącza opcję `wrapscan`

Aby wyświetlić wszystkie opcje instrukcji `set`, należy wykonać polecenie `:set all`.

Tabela na następnej stronie zawiera zestaw podstawowych poleceń edytora `vi`, wystarczający do sprawnej pracy z tym narzędziem.

Polecenia edytora vi

n oznacza liczbę, która może być opcjonalnie wpisana w niektórych poleceniach

Przesuwanie kursora

*n*h	*n* znaków w lewo
*n*j	*n* znaków w dół
*n*k	*n* znaków w górę
*n*l	*n* znaków w prawo
Ctrl+F	o jeden ekran w dół
Ctrl+B	o jeden ekran w górę
Ctrl+D	o pół ekranu w dół
Ctrl+U	o pół ekranu w górę
H	Początek pierwszego wiersza w bieżącym oknie
M	Początek środkowego wiersza w bieżącym oknie
L	Początek ostatniego wiersza w bieżącym oknie
G	Początek ostatniego wiersza pliku
*n*G	Początek *n*-tego wiersza pliku
*n*w	*n* słów do przodu
*n*b	*n* słów wstecz
0	Początek bieżącego wiersza
$	Koniec bieżącego wiersza

Wpisywanie tekstu

i	Rozpoczęcie wpisywania przed bieżącym położeniem kursora
a	Rozpoczęcie wpisywania za bieżącym położeniem kursora
I	Rozpoczęcie wpisywania na początku wiersza
A	Rozpoczęcie wpisywania na końcu wiersza
R	Wpisywanie tekstu w bieżącym położeniu kursora z usunięciem dotychczasowego tekstu do końca wiersza
o	Otwarcie nowego wiersza poniżej bieżącego wiersza
O	Otwarcie nowego wiersza powyżej bieżącego wiersza

Usuwanie tekstu

*n*dd	Usunięcie *n* wierszy
*n*dw	Usunięcie *n* słów
*n*x	Usunięcie *n* znaków
X	Usunięcie znaku przed kursorem
D	Usunięcie tekstu od kursora do końca wiersza

Zmiany

*n*cc	Zmiana n znaków bieżącego wiersza
c$	Zmiana tekstu od kursora do końca wiersza
C	Zmiana tekstu od kursora do końca wiersza
~	Zmiana wielkości bieżącego znaku
J	Połączenie bieżącego wiersza z następnym
u	Anulowanie ostatniej zmiany
.	Powtórzenie ostatniej zmiany
s	Podstawienie tekstu w miejsce bieżącego znaku
S	Podstawienie tekstu w miejsce bieżącego wiersza
:s	Podstawienie w całym tekście :s/tekst1/tekst2/g
&	Powtórzenie ostatniego polecenia :s
*n*yy	Skopiowanie *n* wierszy do bufora
y*n*w	Skopiowanie *n* słów do bufora
p	Wklejenie skopiowanego lub wyciętego tekstu za kursorem
P	Wklejenie skopiowanego lub wyciętego tekstu przed kursorem

Operacje na plikach

ZZ lub :x	Zapis do pliku i zamknięcie edytora
:w	Zapis do pliku
:wq	Zapis do pliku i zamknięcie edytora
:w!	Wymuszony zapis do pliku
:q	Zamknięcie edytora bez zapisu
:q!	Zamknięcie edytora i anulowanie wszystkich zmian
:n	Przejście do edycji następnego pliku
:f *plik*	Zmiana nazwy bieżącego pliku na *plik*
:r *plik*	Wczytanie zawartości pliku w bieżącym położeniu kursora
:!*p*	Wykonanie polecenia powłoki
:r!*p*	Wpisanie wyniku polecenia powłoki w bieżącym położeniu kursora

Powłoki

Powłoka pełni rolę pośrednika między użytkownikiem a jądrem systemu. Jest interpreterem poleceń wpisywanych przez użytkownika. Podczas uruchamiania powłoki odczytywane są odpowiednie pliki konfiguracyjne. Zawarte są w nich definicje zmiennych środowiskowych, nazwy zastępcze poleceń oraz inne polecenia.

Pierwszą powłokę Uniksa napisał Ken Thompson (Bell Laboratories). Był to stosunkowo prosty interpreter poleceń. Polecenia mogły być łączone w dłuższy ciąg za pomocą potoków. Można też było uruchamiać skrypty, ale brakowało zmiennych, sterowania przebiegiem programu i funkcji. W połowie lat 70 rozwijane były dwie powłoki. Jedna z nich, autorstwa Johna Masheya, była rozszerzeniem powłoki Thompsona. Drugą napisał Steve Bourne. Powłoka Bourne'a umożliwiała sterowanie przebiegiem skryptu i zawierała szereg innych usprawnień. Ze względu na brak zgodności między tymi powłokami powołano komisję, której zadaniem był wybór jednej z nich w celu uczynienia z niej standardowej powłoki Uniksa. Wybór padł na powłokę Bourne'a. Ta lub inna, kompatybilna z nią powłoka (`bash`, `ksh` lub `zsh`), dostępna jest w każdym Uniksie. Druga klasa powłok to powłoka C, uruchamiana poleceniem `csh` i jej pochodne, na przykład tcsh. Lista powłok obsługiwanych przez system znajduje się w pliku `/etc/shells`.

Następująca tabela zawiera zestawienie niektórych cech najbardziej popularnych powłok Uniksa. Jest to część comiesięcznego postingu *UNIX shell differences* zamieszczanego w grupie dyskusyjnej Usenet `comp.unix.shells`.

Opcja	**Powłoka**					
	sh	**csh**	**ksh**	**bash**	**tcsh**	**zsh**
Kontrola procesów	N	T	T	T	T	T
Nazwy zastępcze poleceń (aliasy)	N	T	T	T	T	T
Definiowanie funkcji	T(1)	N	T	T	N	T
"Sensowne" przekierowania wejścia/wyjścia	T	N	T	T	N	T
Historia poleceń	N	T	T	T	T	T
Edycja wiersza poleceń	N	N	T	T	T	T
Uzupełnianie nazwy pliku	N	T(1)	T	T	T	T
Możliwość programowania sposobu uzupełniania nazw	N	N	N	N	T	T
Koprocesy	N	N	T	N	N	T
Wbudowane obliczanie wyrażeń arytmetycznych	N	T	T	T	T	T
Zmiana tekstu zgłoszenia powłoki (w łatwy sposób)	N	N	T	T	T	T
Typ składni	sh	csh	sh	sh	csh	sh
Automatyczne sprawdzanie poczty	N	T	T	T	T	T
Obsługa długich list argumentów	T	N	T	T	T	T
Możliwość pominięcia plików startowych użytkownika	N	T	N	T	N	T
Możliwość wskazania pliku startowego	N	N	T	T	N	N

Obsługa opcji `noclobber`	N	T	T	T	T	T
Obsługa zmiennych lokalnych	N	N	T	T	N	T

Objaśnienia:

T Opcja dostępna
N Opcja niedostępna
(1) Opcja nieobecna w pierwotnej wersji powłoki; dziś prawie zawsze w niej dostępna.

Powłoka Bourne'a

W każdej wersji systemu Unix jest dostępna powłoka Bourne'a, wywoływana poleceniem `sh` lub inna wywodząca się z niej powłoka, na przykład `bash` (skrót od słów Bourne Again SHell).

Konfiguracja powłoki Bourne'a jest zdefiniowana w pliku `.profile`. Część definicji może znajdować się w pliku konfiguracyjnym dla całego systemu `/etc/profile`. Systemowy plik konfiguracyjny wykonywany jest przed uruchomieniem pliku `.profile` w katalogu głównym użytkownika. Oto zawartość przykładowego pliku `.profile`:

```
PATH=/sbin:/usr/bin:/usr/local/bin:.
export PATH

# Ustawienie tekstu zgloszenia (ang. prompt)
PS1="`hostname`@`whoami` > "

# Dodatkowe funkcje
ls() { /bin/ls -sF "$@";}
ll() { ls -al "$@";}

# Definicja klawisza usuwającego znaki w wierszu poleceń;
# kombinacja Ctrl+H (^H) odpowiada zwykle klawiszowi
# Backspace
stty erase ^H

# Domyślna maska uprawnień do nowo tworzonych plików
umask 077
```

Tekst od znaku # do końca wiersza jest traktowany jako komentarz. W zmiennej `PATH` poszczególne katalogi są oddzielone średnikiem. Kropka `.` oznacza katalog bieżący. Jeżeli katalog bieżący nie jest częścią zmiennej `PATH`, wywołując program w katalogu bieżącym, należy poprzedzić go kropką i ukośnikiem:

```
./nazwa_programu
```

Ze względu na bezpieczeństwo systemu nie należy jednak umieszczać kropki w zmiennej `PATH`.

Zmienne zdefiniowane w pliku `.profile` obowiązują jedynie w powłoce, do której użytkownik się loguje. W celu przeniesienia definicji na wszystkie powłoki potomne należy dodać polecenie `export`. Innym sposobem przeniesienia definicji z pliku `.profile` jest wykonanie polecenia

```
source .profile
```

lub

```
. ./.profile
```

Aby ułatwić sobie pracę, można zdefiniować własne, prywatne funkcje. W powyższym przykładzie zdefiniowane zostało polecenie `ll`, które jest równoważne wykonaniu polecenia `ls -al`. Aby w poleceniu `stty` prawidłowo wpisać kombinację znaków `^H`, należy w trybie wpisywania w edytorze `vi` najpierw nacisnąć kombinację klawiszy **Ctrl+V**, a następnie nacisnąć **Ctrl+H**. W ostatnim wierszu przykładu argument polecenia `umask` został wybrany tak, by tworzone pliki i katalogi nie dawały żadnych praw dostępu (odczytu, zapisu ani wykonywania) grupie (ang. *group*) ani pozostałym (ang. *other*).

Wśród użytecznych funkcji, jakie można umieścić w pliku `.profile` jest zdefiniowanie tekstu zgłoszenia powłoki (ang. *prompt*) w taki sposób, aby wyświetlał ścieżkę do bieżącego katalogu:

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

Aby wyświetlić wartości wszystkich zmiennych środowiskowych, należy wykonać polecenie `set`. Oto wynik tego polecenia w powłoce bash, wykonanego przez użytkownika `ktos` w systemie Red Hat Linux 8.0:

```
$ set
BASH=/bin/bash
BASH_VERSINFO=([0]="2" [1]="05b" [2]="0" [3]="1" [4]="release" [5]="i686-
pc-linux-gnu")
BASH_VERSION='2.05b.0(1)-release'
COLORS=/etc/DIR_COLORS.xterm
COLORTERM=gnome-terminal
COLUMNS=80
DIRSTACK=()
DISPLAY=:0
EUID=500
GNOME_DESKTOP_SESSION_ID=Default
GROUPS=()
GTK_RC_FILES=/etc/gtk/gtkrc:/home/ktos/.gtkrc-1.2-gnome2
G_BROKEN_FILENAMES=1
HISTFILE=/home/ktos/.bash_history
HISTFILESIZE=10
HISTSIZE=10
HOME=/home/ktos
HOSTNAME=localhost.localdomain
HOSTTYPE=i686
IFS=$' \t\n'
INPUTRC=/etc/inputrc
LANG=en_US.UTF-8
LESSOPEN='|/usr/bin/lesspipe.sh %s'
```

```
LINES=24
LOGNAME=ktos
LS_COLORS='no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01:cd=4
0;33;01:or=01;05;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe=00;32:*.co
m=00;32:*.btm=00;32:*.bat=00;32:*.sh=00;32:*.csh=00;32:*.tar=00;31:*.tgz=00
;31:*.arj=00;31:*.taz=00;31:*.lzh=00;31:*.zip=00;31:*.z=00;31:*.Z=00;31:*.g
z=00;31:*.bz2=00;31:*.bz=00;31:*.tz=00;31:*.rpm=00;31:*.cpio=00;31:*.jpg=00
;35:*.gif=00;35:*.bmp=00;35:*.xbm=00;35:*.xpm=00;35:*.png=00;35:*.tif=00;35
:'
MACHTYPE=i686-pc-linux-gnu
MAIL=/var/spool/mail/ktos
MAILCHECK=60
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/ktos/bin
PIPESTATUS=([0]="0")
PPID=892
PROMPT_COMMAND='echo -ne
"\033]0;${USER}@${HOSTNAME%%.*}:${PWD/#$HOME/~}\007"'
PS1='[\u@\h \W]\$ '
PS2='> '
PS4='+ '
PWD=/home/ktos
SESSION_MANAGER=local/localhost.localdomain:/tmp/.ICE-unix/832
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-com-
ments:monitor
SHLVL=3
SSH_AGENT_PID=848
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
SSH_AUTH_SOCK=/tmp/ssh-XXzcnzaf/agent.832
SUPPORTED=en_US.UTF-8:en_US:en:pl_PL.UTF-8:pl_PL:pl
TERM=xterm
UID=500
USER=ktos
WINDOWID=25165965
XAUTHORITY=/home/ktos/.Xauthority
XMODIFIERS=@im=none
_=set
i=/etc/profile.d/which-2.sh
langfile=/home/ktos/.i18n
mc ()
{
    mkdir -p $HOME/.mc/tmp 2>/dev/null;
    chmod 700 $HOME/.mc/tmp;
    MC=$HOME/.mc/tmp/mc-$$;
    /usr/bin/mc -P "$@" >"$MC";
    cd "`cat $MC`";
    /bin/rm -f "$MC";
    unset MC
}
```

Powłoka C

Powłoka C (polecenie `csh`) powstała w ramach projektu Berkeley UNIX System. Jej autorem jest Bill Joy. Składnia tej powłoki jest pod wieloma względami podobna do składni języka C. Jedną z jej zalet w porównaniu z pierwotną wersją powłoki Bourne'a była kontrola procesów i możliwość definiowania aliasów (skrótów, zastępczych nazw poleceń). Jednak oprócz zalet

`csh` ma również wady. Okazało się, że tworzenie zaawansowanych skryptów jest w niej utrudnione i straciła ona z czasem na popularności.

Konfigurację powłoki C określają pliki `.cshrc` i `.login`. W niektórych wariantach Uniksa używany jest systemowy plik `/etc/csh.login` konfigurujący powłokę dla całego systemu. Plik `.login` jest wykonywany jedynie podczas logowania użytkownika do systemu. Plik `.cshrc` jest wykonywany przy każdym wykonaniu polecenia `csh` (również podczas logowania). Składnia pliku `.cshrc` różni się nieco od składni pliku `.profile`. Wartości zmiennych są nadawane poleceniem `set` lub `setenv`. `set` definiuje wartości dla bieżącej powłoki, a `setenv` dla bieżącej powłoki i wszystkich podpowłok. Zmienne środowiskowe `USER`, `TERM` i `PATH` są automatycznie importowane i eksportowane, na podstawie wartości zmiennych `user`, `term` i `path`. Dlatego można je pominąć przy wykonywaniu polecenia `setenv`. W powłoce C znak ~ (tylda) oznacza katalog główny użytkownika. Na przykład, zapis `~nowak/.cshrc` oznacza plik `.cshrc` w katalogu głównym użytkownika nowak, a `~/.cshrc` jest plikiem należącym do bieżącego użytkownika.

Oto lista domyślnie definiowanych zmiennych powłoki C:

Zmienna	**Znaczenie**
`argv`	Lista argumentów bieżącej powłoki
`cwd`	Katalog bieżący
`history`	Długość listy historii ostatnio wykonywanych poleceń
`home`	Katalog główny użytkownika; rozpoczyna się od wartości zmiennej `$HOME`
`ignoreeof`	Zdefiniowanie tej zmiennej powoduje ignorowanie znaków `EOF` (**Ctrl+D**, oznacza koniec strumienia danych), przekazywanych z terminala
`noclobber`	Zdefiniowanie tej zmiennej zapobiega zapisaniu na nowo już istniejących plików podczas wykonywania poleceń kierujących dane wyjściowe do plików
`noglob`	Zdefiniowanie tej zmiennej zapobiega rozwinięciu nazwy (ang. *filename expansion*) za pomocą znaku uniwersalnego dopasowania (gwiazdka `*`)
`notify`	Wyświetlanie komunikatu o zakończeniu zadania zaraz po jego zakończeniu
`path`	Ścieżka katalogów przeszukiwanych podczas wykonywania poleceń; rozpoczyna się od wartości zmiennej `$PATH`
`prompt`	Ustawia tekst zgłoszenia w wierszu poleceń (domyślnie `%`)
`savehist`	Liczba wierszy zapamiętanych na liście historii poleceń
`shell`	Pełna ścieżka dostępu do pliku wykonywalnego bieżącej powłoki; rozpoczyna się od wartości zmiennej `$SHELL`
`status`	Stan komunikatu `exit` ostatniego z wykonywanych poleceń poleceń (0 – normalne zakończenie, 1 – niepowodzenie)
`term`	Typ terminala, rozpoczyna się od wartości `$TERM`
`user`	Nazwa użytkownika, rozpoczyna się od wartości `$USER`

Oto przykład pliku `.cshrc`:

```
# Definicja ścieżki
set path=(/sbin /usr/bin /usr/local/bin ~/bin )

# Ustawienie podstawowego tekstu zgłoszenia; domyślnie %
set prompt="'hostname'@'whoami' % "
```

```
set noclobber
set ignoreeof
set notify

# Długość historii poleceń;
set history=50

# Przechowanie historii poleceń do następnego zalogowania
savehist=30

# aliasy, czyli polecenia zastępcze
alias h history
alias ll ls –al
alias rm 'rm –i'
alias cp 'cp –i'
alias mv 'mv –i'

# Dzięki tej definicji po zmianie katalogu drukowana
# jest pełna ścieżka do katalogu bieżącego
alias cd 'cd \!*;pwd'

   umask 077
```

Polecenie `alias` definiuje nazwy zastępcze poleceń, zwykle skrócone. Aby sprawdzić, czy nazwa polecenia jest nazwą zastępczą, należy wykonać polecenie `which`, na przykład

```
% which rm
rm:  aliased to rm -i
```

Po dokonaniu zmian w plikach konfiguracyjnych środowisko powłoki można uaktualnić, wykonując polecenie

```
source .cshrc
```

Powłoka Korna

Powłoka `ksh` powstała w 1982 r. w Bell Laboratories. Jej autorem jest David Korn. Składnia powłoki Korna jest zgodna ze składnią powłoki Bourne'a. Zaimplementowano w niej większość usprawnień wprowadzonych w powłoce C i wprowadzono wiele nowych funkcji. Dzięki temu `ksh` jest nie tylko interpreterem poleceń, ale także językiem programowania aplikacji. Skrypty powłoki Korna działają szybciej niż skrypty powłok Bourne'a i C.

Po serii kolejnych usprawnień w 1993 r. powstała wersja `ksh93`, uwzględniająca nowe trendy w ewolucji języków skryptowych. Oto podstawowe zalety powłoki Korna:

- Zgodność z powłoką Bourne'a (skrypty sh działają bez konieczności dokonywania zmian)
- Ulepszona obsługa operacji wejścia/wyjścia (m.in. obsługa koprocesów i polecenia `printf`)
- Możliwość edycji wiersza poleceń

- Szybsze wykonywanie większości skryptów dzięki wbudowaniu wielu podstawowych narzędzi
- Obsługa jednowymiarowych tablic
- Rozszerzona przestrzeń nazw
- Możliwość obsługi interfejsu w różnych językach
- Możliwość definiowania nowych funkcji wbudowanych w języku C i dołączanie ich do środowiska powłoki za pomocą polecenia `builtin` (jest to możliwe w systemach obsługujących ładowanie i łączenie kodu do bieżącego procesu w czasie wykonywania)
- Obsługa działań arytmetycznych na liczbach całkowitych (o każdej możliwej podstawie z przedziału 2 do 64) i na liczbach zmiennoprzecinkowych podwójnej precyzji
- Przetwarzanie łańcuchów znaków w sposób zgodny ze standardem ANSI C
- Możliwość tworzenia bibliotek funkcji do wielokrotnego wykorzystania w różnych aplikacjach

W 2000 r. kod źródłowy i pliki wykonywalne powłoki Korna udostępniono publicznie w witrynie `www.kornshell.com` bez konieczności ponoszenia opłat licencyjnych.

Następująca tabela zawiera niektóre zmienne środowiskowe tej powłoki.

Zmienna	**Znaczenie**
`CDPATH`	Katalogi przeszukiwane w celu znalezienia argumentów polecenia `cd`
`EDITOR`	Edytor używany do edycji tekstu w wierszu poleceń
`ENV`	Plik uruchamiany przy każdym otwarciu powłoki Korna; zwykle `HOME/.kshrc`, ale może to być dowolny plik
`FCEDIT`	Edytor używany przez polecenie `fc`; domyślnie `/bin/ed`
`IFS`	Wewnętrzny separator pól; domyślnymi wartościami są spacja, tabulator lub znak nowego wiersza
`HISTFILE`	Plik, w którym przechowywana jest lista historii poleceń, domyślnie `$HOME/.sh_history`; należy ją ustawić przed uruchomieniem `ksh`
`HISTSIZE`	Liczba poleceń przechowywanych w pliku historii, domyślnie 128; należy ją ustawić przed uruchomieniem `ksh`
`HOME`	Katalog główny użytkownika
`IFS`	Wewnętrzny separator pól
`LOGNAME`	Nazwa, jakiej użytkownik używa podczas logowania do systemu
`MAIL`	Plik, w którym przechowywana jest poczta przychodząca do użytkownika
`MAILCHECK`	Częstość sprawdzania skrzynki pocztowej (w sekundach)
`PATH`	Zawiera nazwy katalogów przeszukiwanych w celu znalezienia plików wykonywalnych dla poleceń powłoki; domyślnie `/usr/bin`
`PS1`	Podstawowy tekst zgłoszenia; domyślnie `$`
`PS2`	Wtórny tekst zgłoszenia; domyślnie >
`PWD`	Bieżący katalog
`TERM`	Typ terminala
`TMOUT`	Czas w sekundach, po jakim powłoka zostaje zamknięta, jeśli nie wpisano żadnego polecenia

Konfiguracja powłoki Korna może być zdefiniowana w trzech plikach. Jeżeli `ksh` jest powłoką domyślną, podczas logowania do systemu odczytywany jest plik `/etc/profile`.

Następnie odczytywany jest plik `$HOME/.profile`. Część definicji można umieścić w pliku wskazywanym zmienną `ENV` (zwykle jest to `$HOME/.kshrc`), odczytywanym przy każdym uruchomieniu `ksh`.

Oto zawartość przykładowego pliku `/etc/profile`:

```
export ENV=$HOME/.kshrc \
       PATH="$HOME/bin":$PATH \
       HISTFILE=$HOME/.sh_history \
       HISTSIZE=100 \
       MAILCHECK=120 \
       PS2="continue> " \
set –o noclobber
set –o ignoreeof
```

Znak `\` oznacza kontynuację wiersza. Opcja `noclobber` chroni istniejące pliki przed zapisaniem na nowo na przykład w wyniku poleceń kopiowania lub zmiany nazwy. Na przykład, jeżeli `plik2` istnieje, następujące polecenie kopiowania zakończy się niepowodzeniem:

```
$ cp plik1 plik2
plik2: file exists
```

Aby wyłączyć blokadę zapisu istniejących plików, należy użyć opcji `+o`:

```
$ set +o noclobber
```

Po włączeniu opcji `ignoreeof` ignorowany jest sygnał `eof`, przekazywany zwykle za pomocą kombinacji klawiszy **Ctrl+D**. Chroni to użytkownika przed przypadkowym zamknięciem powłoki i wylogowaniem z systemu.

Wartości wszystkich opcji polecenia `set` wyświetlane są za pomocą następującego polecenia:

```
$ set -o
```

W pliku `.kshrc` można umieścić pozostałe definicje, które powinny być uwzględnione przy każdym uruchomieniu powłoki Korna. Oto przykład:

```
# Jeżeli zmienne VISUAL lub EDITOR nie są zdefiniowane
# (łańcucha reprezentujący wartość zmiennej ma długość zero),
# definiuj vi jako edytor wiersza poleceń.
if [ -z "$VISUAL" -a -z "$EDITOR" ]; then
  set -o vi
fi
```

Kontrola procesów

Po zalogowaniu użytkownika jądro systemu uruchamia powłokę, która ma interpretować polecenia i łączy ją z terminalem. Podczas wykonywania polecenia powłoki tworzony jest proces potomny powłoki. Ten proces ma łączność z terminalem, dzięki czemu użytkownik może przekazywać dane do procesu i odbierać dane. Po zakończeniu procesu potomnego powłoka odzyskuje łączność z terminalem. Oznaką gotowości powłoki do przyjmowania dalszych poleceń użytkownika jest wyświetlenie tekstu zgłoszenia (ang. *prompt*). Nie wszystkie programy wymagają interakcji z użytkownikiem. Na przykład umieszczenie zawartości trzech plików w innym pliku za pomocą polecenia `cat` nie wymaga interakcji.

Uruchomione procesy mogą być kontynuowane także po wylogowaniu użytkownika z systemu. Zadania odłączone od terminala (nie pobierające danych z terminala ani nie przekazujące do niego danych) nazywane są procesami w tle (ang. *background jobs*). Zadanie połączone z terminalem jest procesem pierwszego planu (ang. *foreground job*). Terminal może być jednocześnie połączony tylko z jednym takim procesem. Zadań działających w tle może być wiele. Zadanie w tle może być przyłączone do terminala i stać się procesem pierwszego planu. Podobnie, proces pierwszego planu można przenieść w tło.

Po uruchomieniu zadania na pierwszym planie nie można wykonywać następnych poleceń aż do zakończenia programu lub umieszczenia go w tle. Aby uruchomić zadanie w tle, należy na końcu polecenia umieścić znak `&` (ampersand), na przykład:

```
$ cat plik1 plik2 plik3 > plik123 &
```

Polecenie `jobs` wyświetla listę procesów uruchomionych w bieżącej powłoce:

```
$ jobs
[1]-  Running                 xpdf artykul.pdf &
[2]+  Running                 xman &
```

W bieżącej powłoce uruchomione zostały dwa procesy w tle: `xpdf`, przeglądarka dokumentów w formacie pdf i `xman`, interfejs graficzny plików pomocy `man`. Ostatni z uruchomionych procesów nazywany jest procesem bieżącym (oznaczony znakiem +), a pozostałe – procesami poprzednimi (znak −). Domyślnie polecenia kontroli zadań odnoszą się do procesu bieżącego. Oto notacja używana w poleceniu `jobs`:

`%`*n*	Proces nr *n*
`%+` lub `%%`	Proces bieżący
`%-`	Proces poprzedni
`%abc`	Proces, którego wywołanie rozpoczyna się znakami `abc`
`%?abc`	Proces, którego wywołanie zawiera znaki `abc`

Aby zawiesić proces pierwszego planu, należy nacisnąć kombinację klawiszy **Ctrl+Z**. Na przykład, jeżeli program `xman` został uruchomiony na pierwszym planie (bez znaku `&`), naciśnięcie **Ctrl+Z** daje następujący wynik:

```
[2]+  Stopped                 xman
```

Wykonywanie wstrzymanego procesu można wznowić na pierwszym planie:

```
$ fg
xman
```

lub w tle:

```
$ bg
[2]+ xman &
```

Domyślnie polecenia `fg` i `bg` odnoszą się do bieżącego procesu. Aby przenieść na pierwszy plan zadanie nr 2, należy wykonać następujące polecenie:

```
$ fg %2
xman
```

Wykonywanie zadania na pierwszym planie można zakończyć, naciskając kombinację klawiszy **Ctrl+C**. Innym sposobem zamykania procesów zarówno pierwszego planu, jak i tła, jest polecenie `kill`:

```
$ kill identyfikator_procesu
```

Identyfikatorem procesu jest PID, wyświetlany poleceniem `ps` lub numer procesu wyświetlony w wyniku polecenia `jobs -l`:

```
$ ps –ef | grep xman
ktos      7734    919  0 14:03 pts/2    00:00:01 xman
ktos      7805    863  0 14:13 pts/0    00:00:01 grep xman

$ jobs –l
[1]-  5091 Running           xpdf artykul.pdf &
[2]+  7734 Running           xman
```

Identyfikator PID procesu `xman` jest równy `7734`. Oto polecenie zamykające ten proces:

```
$ kill 7734
```

Polecenie `kill` nie zawsze daje oczekiwany efekt. Jeżeli proces nie został zakończony, można użyć „najmocniejszej” opcji `–9`:

```
$ kill –9 7734
```

Możliwe jest też zakończenie wielu procesów jednocześnie. Należy wówczas wpisać identyfikatory wszystkich zadań, które mają być zakończone.

Wygodnym narzędziem do przeprowadzania testów jest program uruchamiany poleceniem `yes`, ponieważ nie wykonuje żadnych operacji oprócz drukowania znaku `y` (domyślnie) lub innego ciągu znaków w każdym wierszu na standardowym wyjściu. Aby nie blokować ekranu, lepiej jest skierować wyjście tego procesu do urządzenia `/dev/null`:

```
$ yes > /dev/null
```

W rezultacie dane nie są ani wyświetlane na ekranie, ani zapisywane w pliku. Naciśnięcie kombinacji klawiszy **Ctrl+C** kończy wykonywanie tego procesu. Jeżeli proces ma być jedynie wstrzymany, ale nie zakończony, należy nacisnąć **Ctrl+Z**. Na ekranie pojawi się komunikat

```
[1]+  Stopped                 yes >/dev/null
```

Ten sam komunikat pojawi się po wykonaniu polecenia `jobs`:

```
$ jobs
[1]+  Stopped                 yes >/dev/null
```

Aby umieścić to zadanie w tle, należy wykonać następujące polecenie:

```
$ bg %1
yes >/dev/null &
```

W celu zakończenia procesu można wykonać polecenie `kill`:

```
$ kill %1
```

lub

```
$ kill -9 %1
```

Deskryptory plików i przekierowania

W Uniksie dane mogą być przekazywane do programu z pliku, klawiatury lub innego programu. Dane wynikowe programu mogą być kierowane do pliku, na ekran lub do innego programu. Jest to możliwe dzięki temu, że Unix obsługuje tzw. przekierowania i potoki. Domyślnym źródłem danych wejściowych jest klawiatura (standardowe wejście, ang. *standard input*). Przekierowanie wejścia oznacza pobieranie danych z pliku. Wynik programu (polecenia) jest kierowany zwykle na ekran (standardowe wyjście, ang. *standard output*). Przekierowanie wyjścia oznacza, że odbiornikiem danych jest plik. Dane wyjściowe można skierować do innego programu za pomocą potoku (ang. *pipe*).

Oprócz standardowego wejścia i standardowego wyjścia zdefiniowane jest również standardowe wyjście błędów, do którego kierowane są komunikaty o błędach. Każdy uruchomiony program ma przypisane trzy deskryptory plików:

Deskryptor pliku	**Znaczenie**
`0`	Standardowe wejście; zwykle klawiatura
`1`	Standardowe wyjście; zwykle ekran
`2`	Standardowe wyjście błędów; zwykle ekran

W następującej tabeli przedstawione są różne warianty przekierowań.

Notacja	**Znaczenie**
`> plik`	Standardowe wyjście jest zapisywane do *`pliku`*
`>> plik`	Standardowe wyjście jest dopisywane do *`pliku`*
`program1` `|` *`program2`*	Wynik *`programu1`* jest przekazywany na wejście *`programu2`*
`< plik`	Dane wejściowe są pobierane z *`pliku`*, zamiast ze standardowego wejścia
`<< ciąg_znaków`	tzw. *here document*, pobieranie danych ze standardowego wejścia, aż do napotkania wiersza *`ciąg_znaków`*; w danych wykonywane są podstawienia typu `$`, `` `...` `` i `\`
`<< \ciąg_znaków`	jw., ale podstawianie przez powłokę jest wyłączone

Oto prosty przykład przekierowania wyjścia:

```
$ cat plik1 plik2 > plik3
```

Łączna zawartość plików *`plik1`* i *`plik2`* została zapisana w *`pliku3`*. Jeśli przed wykonaniem tego polecenia *`plik3`* istniał i nie był pusty, poprzednia zawartość została usunięta. Aby dopisać dane do istniejącego pliku, nie usuwając jego wcześniejszej zawartości, należy użyć przekierowania >>.

```
$ cat plik1 >> plik2
```

Możliwe jest także wysłanie do tego samego pliku wyniku wielu poleceń:

```
$ (date; who) > plik
```

Potok służy do przekazywania wyników programu do innego programu:

```
$ who | wc –l
4
```

Polecenie `who` drukuje listę zalogowanych użytkowników, a `wc –l` wyświetla liczbę wierszy przekazanych na wejściu. Wynika stąd, że w momencie wykonania polecenia z ostatniego przykładu w systemie było zalogowanych 4 użytkowników.

Aby pobrać dane wejściowe z pliku, zamiast ze standardowego wejścia, należy wykonać polecenie

```
$ polecenie < plik
```

Oto przykład, w którym małe litery z *pliku1* są zamienione na wielkie litery, a wynik jest zapisany w *pliku2*:

```
$ tr '[a-z]' '[A-Z]' < plik1 > plik2
```

Polecenie dokonuje zamiany małych liter na wielkie. Polecenie

```
$ tr '[a-z]' '[A-Z]'
```

pobiera tekst z klawiatury i zamienia w nim małe litery na wielkie, drukując wynik na ekranie. Bardziej przydatne jest jednak polecenie `tr` z użyciem przekierowania. Tę samą operację można zapisać także w następującej postaci:

```
$ cat plik1 | tr '[a-z]' '[A-Z]' > plik2
```

Przekierowanie << stosuje się w skryptach powłokowych, jeśli dane mają być przetwarzane do momentu pojawienia się określonego ciągu znaków. Łańcuch wyznaczający kres przetwarzania może być wartością zmiennej lub wynikiem polecenia. Przykład zastosowania takiej operacji jest podany w rozdziale *Usługi komunikacyjne*, w skrypcie automatyzującym działanie programu `ftp`. Aby wyłączyć interpretowanie niektórych metaznaków przez powłokę i wykonywanie podstawień tego typu, należy użyć przekierowania w postaci <<\.

Deskryptory plików umożliwiają dokładniejszą kontrolę przekierowań. Aby skierować zarówno wynik polecenia, jak i komunikaty o błędach, do tego samego pliku, należy wykonać polecenie:

```
$ polecenie > plik 2>&1
```

Jeżeli w przykładzie z początku tego rozdziału *plik1* nie istnieje, na ekranie pojawi się komunikat o błędzie:

```
$ cat plik1 plik2 > plik3
cat: plik1: No such file or directory
```

Za pomocą następującego polecenia informacje o błędach można zapisać w tym samym pliku, co wynik:

```
$ cat plik1 plik2 > plik3 2>&1
```

Aby dopisać wynik i komunikaty o błędach do istniejącego pliku, należy wykonać polecenie

```
$ polecenie >> plik 2>&1
```

Deskryptory można też wykorzystać w celu skierowania wyniku i komunikatów o błędach do innego polecenia:

```
$ polecenie 2>&1 | inne_polecenie
```

Wynik polecenia i informacje o błędach mogą być umieszczone w różnych plikach:

```
$ polecenie 1> wynik 2> komunikaty_błędów
```

W tym przykładzie deskryptor 1 można pominąć, ponieważ domyślnie przekierowanie dotyczy standardowego wyjścia. To samo polecenie można zapisać nieco krócej:

```
$ polecenie > wynik 2> komunikaty_błędów
```

W powłoce Bourne'a dostępne są również deskryptory o wyższych numerach, od 3 do 9, które mogą być wykorzystywane do bardziej złożonych operacji przekierowania.

Tworzenie archiwów, kompresja plików

Polecenie	Znaczenie
`ar`	Tworzenie archiwum plików, modyfikowanie i pobieranie plików z archiwum
`compress`	Kompresja pliku za pomocą algorytmu LZW
`uncompress`	Przywrócenie pierwotnej postaci plików skompresowanych poleceniem `compress`
`gzip`	Kompresja pliku za pomocą algorytmu Lempela-Ziva (LZ77)
`gunzip`	Przywrócenie pierwotnej postaci plików skompresowanych poleceniem `gzip`
`tar`	Tworzenie archiwum plików i pobieranie plików z archiwum
`uuencode`	Konwersja pliku do postaci ASCII w celu przesłania go za pomocą narzędzia mail
`uudecode`	Odtworzenie pliku zakodowanego poleceniem `uuencode`

ar

Polecenie `ar` służy do tworzenia archiwów i zarządzania nimi. Najczęściej wykorzystywane jest do tworzenia i uaktualniania bibliotek. Oto przykład utworzenia biblioteki, w której skład wchodzą wszystkie pliki z rozszerzeniem .o z bieżącego katalogu:

```
$ ar cr lib *.o
```

compress

Polecenie `compress` służy do kompresji plików za pomocą algorytmu LZW.

```
$ compress plik
```

W wyniku tego polecenia powstaje plik z rozszerzeniem `.Z`, który można rozkompresować poleceniem `uncompress`:

```
$ uncompress plik.Z
```

gzip

Polecenie `gzip` służy do kompresji plików. Zastosowano w nim algorytm Lempela-Ziva LZ77 (ten sam algorytm jest stosowany w programach `zip` i `pkzip`). Pliki skompresowane tą metodą otrzymują rozszerzenie `.gz`. Stopień kompresji za pomocą tego polecenia jest zwykle większy niż metodą `compress`. Wprawdzie `gzip` i `gunzip` nie są poleceniami standardowego Uniksa (należą do grupy narzędzi GNU), ale są tak powszechnie używane, że trudno byłoby się bez nich obejść. Znaczna większość archiwów publicznie dostępnych w Internecie jest kompresowana za pomocą programu `gzip`.

Kompresowanie pliku:

```
$ gzip plik
```

Przywrócenie pierwotnej postaci pliku:

```
$ gunzip plik.gz
```

tar

Polecenie `tar` tworzy pliki archiwalne. Pliki archiwalne można następnie poddać kompresji. Oto przykład na stworzenie archiwum, w którego skład wchodzi cała zawartość bieżącego katalogu:

```
$ tar cvf archiwum.tar *
```

Katalog, którego zawartość ma być zarchiwizowana, może być podany zarówno w postaci ścieżki względnej lub pełnej:

```
$ tar cvf archiwum.tar ./katalog/*
```

Następny przykład dotyczy rozpakowania archiwum:

```
$ tar xvf archiwum.tar
```

W procesie rozpakowania odtworzona zostaje pełna struktura katalogów. Należy, oczywiście, pamiętać o tym, że rozpakowując archiwum w katalogu, który nie jest pusty, można doznać przykrej niespodzianki. Jeśli nazwy plików są identyczne z istniejącymi już wcześniej w danym katalogu, miejsce dotychczas istniejących plików zajmą pliki z archiwum. Jak sprawdzić zawartość bez rozpakowania? Następujące polecenie wyświetla nazwy katalogów i plików, do których archiwum to zostanie rozpakowane:

```
$ tar -tf archiwum.tar
```

Opcje `cvf` i `xvf` są typowymi opcjami używanymi przy tworzeniu i odtwarzaniu archiwów. Opcja `-v` powoduje wypisanie na standardowym wyjściu nazw plików wchodzących w skład archiwum. Nie jest to konieczne, ale dzięki temu łatwiej sprawdzić poprawność operacji.

uuencode

Polecenie `uuencode` koduje plik binarny do postaci ASCII. Oto jego postać ogólna:

```
$ uuencode plik_wejściowy nazwa > plik_wyjściowy
```

W pierwszym wierszu pliku wyjściowego zapisywana jest informacja o prawach dostępu i nazwie pliku po odkodowaniu poleceniem `uudecode`. Oto przykład kodowania niewielkiego obrazka w formacie gif:

```
$ uuencode obrazek.gif obrazek.gif > obrazek.uu
```

Plik `obrazek.uu` ma następującą postać:

```
begin 644 obrazek.gif
M1TE&.#EA"``*`/?_`/______S/__F?__9O__,___`/_,___,S/_,F?_,9O_,
```

```
M,__,`/^9__^9S/^9F?^99O^9,_^9`/]F__]FS/]FF?]F9O]F,_]F`/\S__\S
MS/\SF?\S9O\S,_\S`/\`__\`S/\`F?\`9O\`,_\``,S__\S_S,S_F<S_9LS_
M,\S_`,S,_\S,S,S,F<S,9LS,,\S,`,R9_\R9S,R9F<R99LR9,\R9`,QF_\QF
MS,QFF<QF9LQF,\QF`,PS_\PSS,PSF<PS9LPS,\PS`,P`_\P`S,P`F<P`9LP`
M,\P``)G__YG_S)G_F9G_9IG_,YG_`)G,_YG,S)G,F9G,9IG,,YG,`)F9_YF9
MS)F9F9F99IF9,YF9`)EF_YEFS)EFF9EF9IEF,YEF`)DS_YDSS)DSF9DS9IDS
M,YDS`)D`_YD`S)D`F9D`9ID`,YD``&;__V;_S&;_F6;_9F;_,V;_`&;,_V;,
MS&;,F6;,9F;,,V;,`&:9_V:9S&:9F6:99F:9,V:9`&9F_V9FS&9FF69F9F9F
M,V9F`&8S_V8SS&8SF68S9F8S,V8S`&8`_V8`S&8`F68`9F8`,V8``#/__S/_
MS#/_F3/_9C/_,S/_`#/,_S/,S#/,F3/,9C/,,S/,`#.9_S.9S#.9F3.99C.9
M,S.9`#-F_S-FS#-FF3-F9C-F,S-F`#,S_S,SS#,SF3,S9C,S,S,S`#,`_S,`
MS#,`F3,`9C,`,S,```#__P#_S`#_F0#_9@#_,P#_``#,_P#,S`#,F0#,9@#,
M,P#,``"9_P"9S`"9F0"99@"9,P"9``!F_P!FS`!FF0!F9@!F,P!F```S_P`S
MS``SF0`S9@`S,P`S````_P``S```F0``9@``,^X``-T``+L``*H``(@``'<`
M`%4``$0``"(``!$```#N``#=``"[``"J``"(``!W``!5``!$```B```1````
M[@``W0``NP``J@``B```=P``50``1```(@``$>[N[MW=W;N[NZJJJHB(B'=W
M=U55541$1"(B(A$1$0```"'Y!`$``"L`+``````(``H```@F`"NL&$APQ88-
?`@L>/$B!X,*%`Q\NK"`1H<&'#@\Z4-BPH,>"`0$`.P``
`
end
```

Istotny jest pierwszy wiersz tego pliku, rozpoczynający się dyrektywą `begin`. Zapisana jest w nim nazwa pliku i uprawnienia, jakie będą przydzielone po odkodowaniu. Po zakodowaniu pliku binarnego do postaci ASCII można go wysłać w treści wiadomości pocztowej:

```
$ mail –s "zakodowany obrazek" julka@domena.pl < obrazek.uu
```

Aby przywrócić pierwotną postać pliku, odbiorca musi zapisać przesyłkę w pliku tak, aby w pierwszym wierszu pliku znajdowała się dyrektywa `begin`, a następnie wykonać polecenie `uudecode`:

```
$ uudecode obrazek.uu
```

Usługi komunikacyjne

Polecenie	Znaczenie
`finger`	Wyświetlenie informacji o użytkowniku
`ftp`	Program transferu plików
`mail`	Prosty program obsługi poczty
`rcp`	Kopiowanie z hosta zdalnego lub w odwrotnym kierunku
`rlogin`	Logowanie do hosta zdalnego
`rsh`	Uruchomienie powłoki na hoście zdalnym
`telnet`	Program do komunikacji w trybie tekstowym
`ssh`	Program komunikacyjny oparty na protokole SSH
`sftp`	Program transferu plików oparty na protokole SSH

`finger`

Polecenie `finger` wyświetla informacje o użytkowniku. Zwykle jest to nazwa użytkownika, imię i nazwisko, nazwa terminala, czas ostatniego logowania, czas bezczynności i inne dane, zależnie od konfiguracji tej usługi. Oto przykład:

```
$ finger ktos
Login: ktos                     Name: (null)
Directory: /home/ktos           Shell: /bin/bash
On since Thu Apr 17 12:17 (CEST) on tty1      22 minutes 31
seconds idle
No mail.
No Plan.
```

W przypadku próby zdalnego skorzystania z tej usługi należy, oczywiście, podać nazwę hosta:

```
$ finger ktos@xyz.com.pl
```

Program `finger` wykorzystuje informacje zapisane w plikach `.plan` i `.project` w katalogu głównym użytkownika. Powiedzmy, że w pliku `.project` umieszczony został następujący tekst:

```
Uczestnicze w projektach ABC, XYZ i Lepsze Jutro.
```

W pliku `.plan` mogłyby się znaleźć następujące informacje:

```
W biurze Pn-Pt od 9 do 17, tel. 1239876, tel. kom. 6xxyyyzzz.
```

Wówczas polecenie `finger ktos` daje następujący wynik:

```
Login: ktos                     Name: (null)
Directory: /home/ktos           Shell: /bin/bash
On since Thu Apr 17 12:17 (CEST) on tty1      22 minutes 31
seconds idle
```

```
No mail.
Project:
Uczestnicze w projektach ABC, XYZ i Lepsze Jutro.
Plan:
W biurze Pn-Pt od 9 do 17, tel. 1239876, tel. kom. 6xxyyyzzz.
```

W celu ochrony systemu przed włamaniami z zewnątrz administrator systemu może zablokować tę usługę.

`ftp`

Usługa `ftp` (ang. *file transfer program*) służy do transferu plików między komputerem lokalnym i hostem zdalnym. Można ją uruchamiać zarówno w trybie interakcyjnym, jak i wsadowym. Aby połączyć się w trybie interakcyjnym, należy wpisać `ftp` *`nazwa_hosta`*:

```
ftp sunsite.icm.edu.pl
```

Jeżeli na komputerze zdalnym uruchomiony jest serwer `ftpd`, pojawi się pytanie o nazwę użytkownika i hasło. Aby zobaczyć listę poleceń dostępnych w sesji `ftp`, wpisz `help` po uruchomieniu programu. W następującej tabeli umieszczone są nazwy instrukcji `ftp` i ich znaczenie.

Polecenie	**Znaczenie**
`!`	Uruchomienie w trybie interakcyjnym powłoki na komputerze lokalnym. Służy do wykonywania poleceń Uniksa bez opuszczania sesji `ftp`, na przykład polecenie `!ls` drukuje zawartość bieżącego katalogu na komputerze lokalnym
`$`	Uruchomienie makra zdefiniowanego wcześniej za pomocą instrukcji `macdef`
`account`	Przekazanie hasła do systemu zdalnego (`account [`*`hasło`*`]`)
`append`	Dołączenie pliku lokalnego do pliku na komputerze zdalnym
`ascii`	Tekstowy (ASCII) tryb transmisji
`bell`	Powoduje, że po zakończeniu każdego transferu pliku uruchamiany jest sygnał dźwiękowy
`binary`	Binarny tryb transmisji
`bye`	Zamknięcie sesji FTP
`case`	Włączenie lub wyłączenie odwzorowania wielkości liter w nazwach plików. Domyślnie mapowanie jest wyłączone. Jeżeli jest włączone, nazwy plików z odległego komputera, składające się wyłącznie z wielkich liter, są mapowane na nazwy pisane małymi literami na komputerze lokalnym.
`cd`	Zmiana katalogu na komputerze zdalnym
`cdup`	Przejście z katalogu bieżącego do katalogu nadrzędnego na komputerze zdalnym
`chmod`	Zmiana uprawnień pliku na komputerze zdalnym
`close`	Zakończenie sesji FTP na serwerze zdalnym
`cr`	Włączenie lub wyłączenie usuwania znaku powrotu karetki podczas transmisji w trybie ASCII
`delete`	Usunięcie pliku na komputerze zdalnym
`debug`	Włączenie trybu debugowania
`dir`	Wyświetlenie zawartości katalogu odległego z opcją zapisu do pliku w

	katalogu lokalnym
`disconnect`	Polecenie równoważne instrukcji `close`
`form`	Ustawienie formatu transferu. Domyślnym formatem jest `file`
`get`	Pobranie pliku z komputera odległego i zapisanie go w systemie lokalnym
`glob`	Włączenie lub wyłączenie rozwinięcia nazw dla poleceń `mdelete`, `mget` i `mput`
`hash`	Włączenie lub wyłączenie drukowania na ekranie symbolu # dla każdego przesłanego bloku danych. Blok danych ma rozmiar 1024 bajtów
`help`	Wyświetla krótką informacją dotyczącą danego polecenia (`help` *`polecenie`*). Instrukcja `help` bez parametrów powoduje drukowanie nazw wszystkich poleceń `ftp`
`idle`	Ustawia licznik bezczynności na serwerze zdalnym na określoną liczbę sekund (`idle` *`liczba_sekund`*). Pominięcie liczby sekund powoduje wyświetlenie bieżącej wartości tego parametru
`lcd`	Zmiana katalogu bieżącego na komputerze lokalnym. Pominięcie nazwy katalogu powoduje przejście do katalogu głównego użytkownika
`ls`	Wyświetlenie zawartości katalogu na komputerze zdalnym
`macdef`	Definicja makro
`mdelete`	Usunięcie plików na komputerze zdalnym
`mdir`	Działanie podobne do polecenia `dir` z tą różnicą, że można określić wiele plików z komputera zdalnego
`mget`	Pobieranie wielu plików z wykorzystaniem opcji rozwijania nazw
`mkdir`	Utworzenie katalogu na komputerze zdalnym
`mls`	Zapisanie nazw plików z komputera zdalnego do pliku lokalnego
`mode`	Ustawienie trybu transferu plików. Domyślnym trybem jest `stream`
`modtime`	Wyświetlenie czasu ostatniej modyfikacji pliku z komputera zdalnego
`mput`	Przesyłanie wielu plików do komputera zdalnego z wykorzystaniem opcji rozwijania nazw
`newer`	Pobranie pliku jedynie wtedy, kiedy czas ostatniej modyfikacji pliku zdalnego jest późniejszy niż czas modyfikacji pliku lokalnego. Jeżeli plik lokalny, na który wskazuje ta instrukcja, nie istnieje, plik zdalny jest traktowany jako nowszy i zostaje pobrany
`nlist`	Drukowanie listy plików z katalogu hosta zdalnego. W przypadku braku nazwy pliku lokalnego, w którym te dane mają być zapisane, lista jest wyświetlana na ekranie
`nmap`	Włączenie lub wyłączenie mapowania nazw plików
`ntrans`	Włączenie lub wyłączenie mechanizmu translacji znaków w nazwach plików
`open`	Ustanowienie połączenia z określonym serwerem FTP (opcjonalnie można podać też numer portu)
`passive`	Włączenie lub wyłączenie trybu pasywnego. Jeżeli tryb pasywny jest włączony, klient FTP przesyła polecenie `PASV` (zamiast polecenia `PORT`) dla każdego przekazu danych. Polecenie `PASV` oczekuje przekazania przez serwer FTP numeru portu, przez który dane mają być przesłane. Serwer nasłuchuje na tym porcie, a klient próbuje się z tym portem połączyć. Tryb pasywny jest przydatny podczas komunikacji za pośrednictwem routera lub hosta wpływającego na kierunek przesyłu danych.
`prompt`	Włączenie lub wyłączenie pytania o potwierdzenie transferu poszczególnych plików podczas przesyłania wielu plików. Jeżeli ta opcja jest wyłączona, polecenia `mget`, `mput` lub `mdelete` są wykonywane bez pytania o

	potwierdzenie
`proxy`	Wykonanie polecenia `ftp` na połączeniu kontrolnym. Ta instrukcja umożliwia jednoczesną łączność z dwoma zdalnymi serwerami FTP w celu przesłania plików z jednego do drugiego
`put`	Umieszczenie pliku lokalnego na komputerze zdalnym
`pwd`	Wyświetlenie nazwy katalogu bieżącego na komputerze zdalnym
`quit`	Polecenie równoważne `bye`
`quote`	Parametry tego polecenia są przesyłane bez żadnej modyfikacji do zdalnego serwera FTP
`recv`	Polecenie równoważne `get`
`reget`	Pobieranie plików z komputera zdalnego. Jeżeli jednak plik lokalny już istnieje i ma mniejszy rozmiar niż plik z komputera zdalnego, transfer pliku jest kontynuowany od miejsca, w którym najprawdopodobniej transmisja została przerwana
`rename`	Zmiana nazwy pliku na komputerze zdalnym
`reset`	Opróżnienie kolejki odpowiedzi w celu ponownej synchronizacji sekwencji poleceń i odpowiedzi
`restart`	Ponowne uruchomienie polecenia `get` lub `put` w miejscu wskazanym przez znacznik. W systemach UNIX znacznik jest zwykle przesunięciem o określoną liczbę bajtów
`rhelp`	Polecenie skierowane do komputera zdalnego w celu uzyskania informacji o poleceniach `ftp`
`rstatus`	Wyświetlenie informacji o stanie komputera zdalnego lub stanie pliku na komputerze zdalnym
`rmdir`	Usunięcie katalogu na komputerze zdalnym
`runique`	Włączenie lub wyłączenie zapisu plików w systemie lokalnym pod unikalnymi nazwami
`send`	Polecenie równoważne `put`
`sendport`	Włączenie lub wyłączenie polecenia `PORT`. Domyślnie program `ftp`, łącząc się z komputerem zdalnym, wykorzystuje polecenia `PORT`. Użycie poleceń `PORT` może zapobiec opóźnieniom podczas transferu wielu plików. Jeżeli polecenie `PORT` nie działa poprawnie, program `ftp` wykorzystuje domyślny port tej usługi
`site`	Parametry tego polecenia są przesyłane do serwera FTP bez żadnej modyfikacji
`size`	Wyświetlenie rozmiaru pliku znajdującego się na komputerze zdalnym
`status`	Wyświetlenie bieżącego statusu sesji `ftp`
`struct`	Ustawienie struktury transferu plików. Domyślnie jest nią `stream`
`system`	Wyświetlenie informacji o rodzaju i wersji systemu operacyjnego komputera zdalnego
`sunique`	Włączenie lub wyłączenie zapisu plików na komputerze zdalnym pod unikalnymi nazwami.
`trace`	Włączenie lub wyłączenie śledzenia pakietów
`type`	Ustawienie informacji o trybie transferu plików. Wykonanie tego polecenia bez parametrów wyświetla bieżący tryb transferu.
`umask`	Ustawienie domyślnej maski dla nowo tworzonych plików na komputerze zdalnym
`user`	Uwierzytelnienie użytkownika na komputerze zdalnym
`verbose`	Włączenie lub wyłączenie trybu opisowego. W tym trybie wyświetlane są

	wszystkie odpowiedzi serwera FTP
`?`	Polecenie równoważne `help`

W przypadku powtarzających się połączeń `ftp` można zrezygnować z trybu interakcyjnego i stworzyć skrypt automatyzujący sesję `ftp`. Następujący przykład dotyczy sesji `ftp` z hostem `sunsite.icm.edu.pl`.

Oto zawartość pliku:

```
#!/bin/sh
# Uruchom sesję ftp; wykonywane są wszystkie polecenia
# aż do znacznika KONIEC_POLECEN_FTP
ftp -n sunsite.icm.edu.pl << KONIEC_POLECEN_FTP
user anonymous nowak@abc.pl

# Przejdź do wskazanego katalogu
cd ./pub/gnu

# Włącz tryb binarny transferu plików
binary

# Pobierz plik jedynie wówczas, gdy data utworzenia
# lokalnego pliku jest wcześniejsza, niż data na serwerze ftp
newer plik

# Zamknij połączenie
bye
KONIEC_POLECEN_FTP
```

W przypadku transferu plików znajdujących się w różnych katalogach serwera warto byłoby umieszczać je w odpowiednich katalogach na komputerze lokalnym. Wówczas polecenie `newer` (lub polecenie `get`) mogłoby mieć następującą postać:

`newer` */ścieżka/plik* `./sunsite/`*ścieżka/plik*

lub

`newer` */ścieżka/plik* `./sunsite.icm.edu.pl` */ścieżka/plik*

Innym sposobem automatyzacji nawiązywania połączeń `ftp` jest zapisanie podstawowych parametrów połączeń w pliku `.netrc`, znajdującym się w katalogu głównym użytkownika (wskazywanym przez zmienną `HOME`). W `.netrc` można zapisać następujące informacje:

```
machine nazwa
default
login nazwa
password ciąg_znaków
account ciąg_znaków
macdef nazwa
```

Oto przykład pliku `.netrc`:

```
machine rose.man.poznan.pl login jamesbond password
DiamondsR4ever

machine zezowate.szczescie.com
login piszczyk

default login anonymous password jamesbond@agent007.com
```

Zapisywanie prywatnego hasła w pliku tekstowym nie jest, oczywiście, bezpiecznym rozwiązaniem i lepiej z tej opcji nie korzystać, ograniczając się do podania nazwy użytkownika. Niewielka to wprawdzie automatyzacja, jeśli polega tylko na wpisaniu nazwy użytkownika (sesja `ftp` jest uruchamiana poleceniem `ftp` *`nazwa_węzła`* lub `ftp` *`adres_IP`*), ale czasem może być przydatna. Aby uniknąć ryzyka podsłuchiwania sesji `ftp`, należy używać narzędzi obsługujących protokół SSH lub systemu uwierzytelniania Kerberos.

Nawiązywanie łączności z serwerami przyjmującymi logowanie na konto `anonymous` nie jest obciążone tą wadą. W tym przykładzie najbardziej przydatny jest ostatni wiersz. Dyrektywa `default` oznacza, że próby połączeń z węzłami, których nazwa nie znalazła się w tym pliku w poprzednich wierszach, traktowane będą jak połączenia z kontem `anonymous`, przy których w roli hasła wysyłany jest adres poczty elektronicznej użytkownika. Komputer po drugiej stronie nie sprawdza jednak prawdziwości przekazanego adresu. Oczywiście, deklaracje połączeń ftp na konto `anonymous` mogą być wpisywane również za pomocą dyrektywy `machine` *`nazwa`*.

`mail`

Program `mail` służy do obsługi poczty. W następującym przykładzie `root` wykonuje polecenie `mail`. Pojawia się komunikat, że w skrzynce pocztowej są 3 wiadomości i wyświetlana jest krótka informacja o każdej z nich, m.in. data najdeścia wiadomości. Po wykonaniu polecenia `help` na ekranie drukowana jest krótka informacja o poleceniach programu pocztowego. Aby wyjść z programu bez dokonywania zmian w pliku pocztowym, należy wykonać polecenie `x`. Jeżeli zmiany mają być zachowane, należy wykonać polecenie `quit`. Dokładny opis tego programu obsługi poczty jest dostępny w pomocy podręcznej (po wykonaniu polecenia `man mail`).

```
[root@localhost root]# mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/spool/mail/root": 3 messages 3 new
>N  1 root@localhost.local  Fri Mar  7 22:19  31/1007  "LogWatch for
localhos"
 N  2 root@localhost.local  Sun Mar 30 18:39  31/1010  "LogWatch for
localhos"
 N  3 root@localhost.local  Fri Apr  4 18:41  31/1007  "LogWatch for
localhos"
& help
    Mail   Commands
t <message list>       type messages
n                      goto and type next message
e <message list>       edit messages
f <message list>       give head lines of messages
```

```
d <message list>        delete messages
s <message list> file         append messages to file
u <message list>        undelete messages
R <message list>        reply to message senders
r <message list>        reply to message senders and all recipients
pre <message list>            make messages go back to /usr/spool/mail
m <user list>                 mail to specific users
q                       quit, saving unresolved messages in mbox
x                       quit, do not remove system mailbox
h                       print out active message headers
!                       shell escape
cd [directory]                chdir to directory or home if none given

A <message list> consists of integers, ranges of same, or user names
separated
by spaces.  If omitted, Mail uses the last message typed.

A <user list> consists of user names or aliases separated by spaces.
Aliases are defined in .mailrc in your home directory.
& x
[root@localhost root]#
```

Wiadomość można wysłać w trybie interakcyjnym przedstawionym powyżej lub w następujący sposób:

```
$ mail –s "temat wiadomosci" ktos@xyz.com.pl < plik_tekstowy
```

Tekst w cudzysłowie wpisany po opcji `–s` pojawi się w wierszu subject u odbiorcy przesyłki. Przekierowanie < powoduje, że zamiast ze standardowego wejścia, program przyjmuje tekst z pliku *plik_tekstowy*.

Aby zmienić domyślną konfigurację programu mail, należy umieścić odpowiednie definicje w pliku `.mailrc` w katalogu głównym użytkownika. Można, na przykład, zdefiniować aliasy (nazwy zastępcze) zarówno dla pojedynczego odbiorcy, jak i grup adresatów. Oto przykład pliku `.mailrc`:

```
# Ukośnik \ oznacza kontynuowanie wiersza.
# Umieszczenie go jest konieczne, ponieważ adresy aliasa
# muszą znajdować się w tym samym wierszu.
alias praca nowak@firmax.com.pl kowalski@firmax.com.pl \
            dyrektor@firmax.com.pl
alias znajomi ewa@xyz.com.pl piotr@zyx.pl
# aliasy mogą być tworzone z innych aliasów
alias wszyscy praca znajomi jamesbond@agent007.com
```

Polecenie wysłania wiadomości do wszystkich osób, których adresy należą do aliasu `praca` ma następującą postać:

```
$ mail –s "wazna sprawa" praca < plik_tekstowy
```

`rcp`, `rlogin`, `rsh`

Polecenia `rcp`, `rlogin` i `rsh` służą do nawiązywania połączeń z hostem zdalnym bez konieczności uwierzytelniania użytkownika za pomocą hasła. `rcp` kopiuje pliki między hostem lokalnym a hostem zdalnym, `rlogin` loguje użytkownika do hosta zdalnego, a `rsh` wykonuje polecenie na hoście zdalnym.

Domyślnie nazwa użytkownika na komputerze zdalnym jest identyczna z nazwą na komputerze lokalnym. Programy rcp, rlogin i rsh wykorzystują definicje z pliku `.rhosts`. Zawarte są w nim adresy hostów, z którymi użytkownik może się łączyć bez podawania hasła. Taki sposób nawiązywania połączeń jest bardzo złym rozwiązaniem, ponieważ stanowi bardzo duże zagrożenie dla bezpieczeństwa systemu – wystarczy, by intruz włamał się na jeden komputer, by mógł następnie bezproblemowo wejść na konto należące do innego systemu. Jeżeli plik `.rhosts` znajduje się w katalogu głównym użytkownika, należy go jak najszybciej usunąć.

W systemie uwierzytelniania Kerberos można korzystać z narzędzi `rcp`, `rlogin` i `rsh` w taki sposób, aby połączenia były szyfrowane i nie narażone na sieciowy podsłuch (Zobacz np. R.J. Hontanon, *Bezpieczeństwo systemu Linux*, Mikom 2002).

`telnet`

Polecenie `telnet` uruchamia klienta protokołu TELNET. Oto ogólna postać tego polecenia:

`telnet [`*`opcje`*`] [`*`host`* `[`*`port`*`]]`

Domyślnie protokół TELNET wykorzystuje port 23. Oto przykład, w którym nawiązane jest połączenie z hostem `rose.man.poznan.pl`:

```
$ telnet sunsite.icm.edu.pl
Trying 193.219.28.2...
telnet: Unable to connect to remote host: Connection refused
```

Taka odpowiedź staje się regułą w Internecie, ponieważ w tym protokole dane nie szyfrowane. Bezpiecznym alternatywą TELNET-u jest protokół SSH, nie będący wprawdzie częścią systemu Unix, ale szeroko rozpowszechniony. Programy komunikacyjne oparte na SSH są bardzo często instalowane w dzisiejszych wariantach Uniksa.

Jeżeli port 23 nie jest zablokowany, host odpowiada pytaniem o nazwę użytkownika i hasło. Jeżeli są prawidłowe, host udostępnia konto użytkownika.

Można jednak polecenia `telnet` użyć do połączeń z tymi portami hosta, które nie zostały zablokowane. Na przykład usługa WWW działa zwykle na porcie 80 i z tym portem można się połączyć:

```
$ telnet sunsite.icm.edu.pl 80
Trying 193.219.28.2...
Connected to sunsite.icm.edu.pl.
Escape character is '^]'.
```

Dalsza komunikacja z hostem zdalnym wymaga znajomości poleceń protokołu HTTP. Oto przykład wysłania wiadomości pocztowej za pomocą serwera SMTP, działającego na porcie

25. Użytkownik `jan.kowalski@abc.org` wysyła wiadomość do użytkownika `adam.nowak@firma.com`. Odpowiedzi serwera są ujęte w cudzysłów (wszystkie adresy są fikcyjne):

```
telnet serwer.smtp 25

"220 serwer.smtp ESMTP Mailserver"

helo abc.org

"250 OK"

mail from: jan.kowalski@abc.org

"250 OK"

rcpt to: adam.nowak@firma.com

"250 OK"

data

"354 End data with <CR><LF>.<CR><LF>

Tekst wiadomości
...
Ostatni wiersz wiadomości
.
"250 Ok.: queued as C6D8F146C00"

quit
```

Na zakończenie wiadomości należy nacisnąć klawisz **Enter**, następnie klawisz **.** (kropka) i ponownie **Enter**, po czym wykonać polecenie `quit`. Te same czynności można wykonać za pomocą następującego skryptu:

```
#!/bin/sh
{    echo 'helo '$4
     echo 'mail from: <'$3'@'$4'>'
     echo 'rcpt To: '$1'@'$2
     echo 'data'
     cat  plik
     echo '.'
     echo 'quit'
}    | telnet $5 25
```

Skrypt należy uruchomić w następujący sposób:

```
$ ./send.sh adam.nowak firma.com jan.kowalski abc.org serwer.smtp
```

Telnet można wykorzystać również w celu odczytania poczty z serwera POP3 na porcie 110:

```
telnet pop3.nazwa.serwera 110

"+OK"

user ktos

"+OK"

pass mojehasło

"+OK"
```

Teraz można odczytać pocztę. Oto podstawowe polecenia serwera POP3:

Polecenie	Znaczenie
`stat`	Wyświetlenie liczby wiadomości w skrzynce pocztowej na serwerze i łącznego rozmiaru wszystkich wiadomości w bajtach
`list`	Wyświetlenie rozmiaru każdej wiadomości w bajtach
`retr` *n*	Wyświetlenie wiadomości nr *n*
`dele` *n*	Usunięcie wiadomości nr *n*
`rset`	Ustawienie stanu wszystkich wiadomości na `unread` (nie odczytane)
`quit`	Zakończenie połączenia z serwerem

Aby zakończyć odczytywanie poczty, należy wpisać polecenie `quit`.

Trzeba pamiętać o tym, że `telnet` nie ma żadnej ochrony przed próbami podsłuchu podczas przesyłania danych przez sieć i administratorzy coraz częściej blokują dostęp do tej usługi. W internetowych archiwach dostępne są gotowe programy umożliwiające prowadzenie takiego podsłuchu. Z tego względu zaleca się wykorzystywanie programów komunikacyjnych szyfrujących transmisję danych za pomocą algorytmów odpornych na złamanie prostymi metodami.

`ssh`, `sftp`

Protokół SSH jest oparty na architekturze klient-serwer. Wszystkie połączenia inicjowane są po stronie klienta. Sesja komunikacyjna jest szyfrowana kluczem 128-bitowym. Chronione są nazwa użytkownika, hasło oraz dane przesyłane w obu kierunkach. Istnieje wiele implementacji protokołu SSH. Wśród użytkowników programów *open source* popularna jest wersja OpenSSH (strona główna projektu: `www.openssh.org`), rozpowszechniana na licencji GPL.

Aby zalogować się w trybie interakcyjnym do hosta *komp.abcde.com*, użytkownik *ktos* powinien wykonać następujące polecenie (jest to bezpieczny odpowiednik sesji telnet):

```
$ ssh -l ktos komp1.abcde.com
ktos@komp1.abcde.com's password:
Last login: Tue May  14 15:32:27 2003 from komp.lokalny.com
komp1 >
```

Po wpisaniu poprawnego hasła użytkownik jest zalogowany na serwerze.

W celu uruchomienia sesji transferu plików w bezpieczny sposób, należy wykonać następujące polecenie:

```
$ sftp ktos@komp1.abcde.com
connecting to komp1.abcde.com...
ktos@komp1.abcde.com's password:
sftp>
```

Najważniejsze polecenia sesji `sftp` są podobne do poleceń `ftp`. Aby uzyskać pomoc, należy wykonać polecenie `help`.

Inne polecenia

Polecenie	**Znaczenie**
`banner`	Drukowanie powiększonego tekstu na standardowym wyjściu
`bc`	Kalkulator działający w arytmetyce dowolnej precyzji
`cal`	Wyświetlenie kalendarza
`clear`	Czyszczenie ekranu
`xargs [`*`polecenie`*`]`	Wykonanie *`polecenia`* z podanymi argumentami; pozostałe argumenty są odczytywane ze standardowego wejścia; domyślnie wykonywane jest polecenie `echo`

banner

Polecenie `banner` wyświetla w powiększeniu napis podany jako argument polecenia:

```
$ banner ola

           XX
            X
  XXXX      X     XXXXX
 X    X     X         X
 X    X     X    XXXXXX
  XXXX    XXXXX  XXXX XX
```

bc

Kalkulator `bc` służy do wykonywania obliczeń arytmetycznych z dowolną precyzją. Składnia poleceń jest podobna do składni języka C. Dostępna jest biblioteka funkcji matematycznych (opcja `-l`). Przetwarzany jest kod obliczeń ze wszystkich plików podanych w wierszu poleceń. Po zakończeniu przetwarzania plików kalkulator przyjmuje polecenia ze standardowego wejścia.

Powiedzmy, że zadanie polega na obliczeniu wartości funkcji `exp(-x^2)` w przedziale od `-1` do `1`. W pliku *`obliczenia`* zapisane są następujące polecenia:

```
/* Mały program do obliczenia wartosci funkcji */
x0=0                /* Początek przedziału */
x1=1                /* Koniec przedziału */
imax=5              /* Liczba iteracji */
dx=(x1-x0)/imax     /* Przyrost zmiennej x w jednej iteracji */

/* definicja funkcji */
define f(x) {
  auto v,w          /* Definicja zmiennych lokalnych */
  w=x^2
  v=e(-w)
  return (v)        /* Zwracana jest wartość zmiennej v */
}
```

```
for (i=0; i<=imax; i++) {
  x=x0+i*dx         /* Bieżąca wartość zmiennej x */
  y=f(x)
  "x="; x; "y="; y  /* Drukowanie wartości zmiennych x i y */

/* Następna instrukcja ma na celu wydrukowanie pustego
wiersza. Między pierwszym a drugim znakiem cudzysłowu
wciśnięty został klawisz Enter. Dzięki temu na wyjściu pojawia
się pusty wiersz między kolejnymi parami wartości x i y */
  "
"
}
```

Uruchomienie kalkultora daje następujący wynik:

```
$ bc -l obliczenia
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software
Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
x=0
y=1.00000000000000000000

x=.20000000000000000000
y=.96078943915232320943

x=.40000000000000000000
y=.85214378896621133845

x=.60000000000000000000
y=.69767632607103105720

x=.80000000000000000000
y=.52729242404304855724

x=1.00000000000000000000
y=.36787944117144232159
```

W celu zwiększenia precyzji obliczeń należy w kodzie programu użyć instrukcji `scale=`*n*, gdzie *n* oznacza liczbę miejsc po przecinku. Domyślną wartością tego parametru jest 20. Po umieszczeniu instrukcji `scale=40` na początku programu otrzymujemy następujący wynik:

```
x=0
y=1.0000000000000000000000000000000000000000

x=.2000000000000000000000000000000000000000
y=.9607894391523232094392106913232458860279

x=.4000000000000000000000000000000000000000
```

```
y=.852143788966211338456346981468562659774l

x=.600000000000000000000000000000000000000000
y=.697676326071031057209129263838160330228 7

x=.800000000000000000000000000000000000000000
y=.527292424043048557243694608566314482486 1

x=1.000000000000000000000000000000000000000000
y=.367879441171442321595523770161460867445 8
```

Aby zakończyć działanie programu, należy wpisać wykonać polecenie `quit` lub nacisnąć kombinację klawiszy **Ctrl+D**.

Podstawowe funkcje matematyczne oznaczone są następująco:

Notacja	**Znaczenie**
`s(x)`	sinus (w radianach)
`c(x)`	cosinus
`a(x)`	arcus tangens
`l(x)`	logarytm naturalny
`e(x)`	liczba `e` do potęgi `x`
`j(n,x)`	funkcja Bessela rzędu `n` (`n` jest liczbą całkowitą) argumentu `x`

Oto przykład obliczenia liczby `pi` za pomocą `bc`.

```
$ pi=$(echo "scale=10; 4*a(1)" | bc -l)
$ echo $pi
3.1415926532
```

W pierwszym poleceniu zmiennej pi przyporządkowany jest wynik wyrażenia w nawiasie. Porównując otrzymany wynik z tablicami matematycznymi, `3.14159 26535`, zauważymy, że ostatnia cyfra jest obarczona jest błędem. Aby zmniejszyć ten błąd lub go uniknąć, należałoby zwiększyć liczbę cyfr znaczących, a uzyskany wynik zaokrąglić do potrzebnej liczby cyfr znaczących.

Pełna dokumentacja kalkulatora bc jest umieszczona w pliku pomocy podręcznej (polecenie `man bc`).

`cal`

Polecenie `cal` drukuje fragment kalendarza na standardowym wyjściu:

```
$ cal
      May 2003
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
```

```
25 26 27 28 29 30 31
```

Domyślnie drukowany jest kalendarz na bieżący miesiąc. Aby otrzymać kalendarz na dowolny miesiąc lub rok, należy wpisać w wierszu poleceń numer miesiąca lub roku:

```
$ cal 1 2004
    January 2004
Su Mo Tu We Th Fr Sa
              1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

$ cal 2003
                                2003

       January               February                March
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
          1  2  3  4                      1                      1
 5  6  7  8  9 10 11    2  3  4  5  6  7  8    2  3  4  5  6  7  8
12 13 14 15 16 17 18    9 10 11 12 13 14 15    9 10 11 12 13 14 15
19 20 21 22 23 24 25   16 17 18 19 20 21 22   16 17 18 19 20 21 22
26 27 28 29 30 31      23 24 25 26 27 28      23 24 25 26 27 28 29
                                              30 31

        April                  May                   June
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
       1  2  3  4  5                   1  2  3    1  2  3  4  5  6  7
 6  7  8  9 10 11 12    4  5  6  7  8  9 10    8  9 10 11 12 13 14
13 14 15 16 17 18 19   11 12 13 14 15 16 17   15 16 17 18 19 20 21
20 21 22 23 24 25 26   18 19 20 21 22 23 24   22 23 24 25 26 27 28
27 28 29 30            25 26 27 28 29 30 31   29 30

        July                  August               September
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
       1  2  3  4  5                1  2           1  2  3  4  5  6
 6  7  8  9 10 11 12    3  4  5  6  7  8  9    7  8  9 10 11 12 13
13 14 15 16 17 18 19   10 11 12 13 14 15 16   14 15 16 17 18 19 20
20 21 22 23 24 25 26   17 18 19 20 21 22 23   21 22 23 24 25 26 27
27 28 29 30 31         24 25 26 27 28 29 30   28 29 30
                       31

       October               November              December
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
          1  2  3  4                      1           1  2  3  4  5  6
 5  6  7  8  9 10 11    2  3  4  5  6  7  8    7  8  9 10 11 12 13
12 13 14 15 16 17 18    9 10 11 12 13 14 15   14 15 16 17 18 19 20
19 20 21 22 23 24 25   16 17 18 19 20 21 22   21 22 23 24 25 26 27
26 27 28 29 30 31      23 24 25 26 27 28 29   28 29 30 31
                       30
```

Można też drukować kalendarz na kilka miesięcy. Na przykład, aby otrzymać kalendarz na pierwszą połowę roku, należy wykonać polecenie `cal 1-6`. Aby wyświetlić poniedziałek jako pierwszy dzień tygodnia, należy użyć opcji `-m`:

```
$ cal -m
      May 2003
Mo Tu We Th Fr Sa Su
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

xargs

Polecenie `xargs` umożliwia przetworzenie większej liczby argumentów poleceń, niż byłoby to normalnie możliwe. Dość często `xargs` jest zestawiane z poleceniem `find` w celu wykonania operacji na wielu plikach. Oto przykład. Aby wyszukać pliki zawierające kod źródłowy języka C i przenieść je do bieżącego katalogu, można wykonać następujące polecenie:

```
$ find ./ -type f -name "*.c" -print -exec mv {} . \;
```

Jeżeli liczba znalezionych plików jest duża, mogą pojawić się trudności. Zamiast opcji `-exec`, lepiej użyć polecenia `xargs`:

```
$ find ./ -type f -name "*.c" -print | xargs mv {} . ./
```

Za pomocą opcji `-l` można ograniczyć liczbę argumentów przekazywanych jednorazowo przez `xargs` do polecenia `mv` (w tym przykładzie argumenty są przekazywane grupami, po 32 argumenty w grupie):

```
$ find ./ -type f -name "*.c" -print | xargs -l32 mv {} . ./
```

`xargs` można także użyć w celu umieszczenia wyniku kilku poleceń w jednym wierszu:

```
$ (logname; date) | xargs
```

Wyrażenia regularne

Nazwa „wyrażenia regularne” (ang. *regular expressions*) oznacza notację stosowaną w celu wyszukiwania i dopasowywania ciągów znaków. Narzędzia Uniksa przeznaczone do przetwarzania tekstu umożliwiają zwykle wyszukiwanie tzw. wzorców zamiast dokładnych łańcuchów znaków. W ich skład wchodzą zwykłe znaki (litery, cyfry) i znaki specjalne (metaznaki). Oto metaznaki używane w wyrażeniach regularnych:

```
\   ^   .   [   ]   |   (   )   *   +   ?
```

Są trzy rodzaje wyrażeń regularnych:

- Metaznaki pozycyjne
- Zbiory znaków
- Modyfikatory oznaczające liczbę powtórzeń ostatniego wyrażenia

Oto podstawowe rodzaje wyrażeń regularnych:

- Zwykły znak (nie będący metaznakiem), np. `a`, który pasuje do samego siebie
- Sekwencja oznaczająca znak specjalny, np. `\t` oznacza tabulator
- Cytowany metaznak, np. `\*`, pozbawiony w ten sposób specjalnego znaczenia
- Znak `^` oznaczający początek łańcucha
- Znak `$` oznaczający koniec łańcucha
- Kropka `.` oznaczająca dowolny pojedynczy znak
- Klasa znaków, np. `[ABC]` odpowiada dowolnemu spośród znaków `A`, `B` lub `C`. Klasy znaków mogą zawierać skróty, np. `[A-Za-z]` odpowiada dowolnej pojedynczej literze
- Zaprzeczenie klasy znaków, np. `[^0-9]` odpowiada dowolnemu znakowi, oprócz cyfry

Wyrażenie regularne, składające się z grupy znaków zawartej w nawiasach kwadratowych, nazywane jest klasą znaków. Odpowiada ono dowolnemu pojedynczemu znakowi spośród znaków zawartych w nawiasach. Na przykład, `[abcd]` odpowiada `a`, `b`, `c` lub `d`.

Zamiast wypisywać kilka następujących po sobie znaków, można użyć notacji skróconej, np. `[a-d]`, `[0-9]`, `[A-Da-d][0-9]`. Ostatni z tych trzech przykładów oznacza jedną literę, wielką lub małą, z zakresu od `a` do `d`, po której następuje dowolna cyfra. Zapis `[+-]` lub `[-+]` oznacza znak + lub -.

Wyrażenie, w którym pierwszym znakiem po nawiasie kwadratowym `[` jest znak `^`, oznacza domknięcie klasy znaków. Do domknięcia należą wszystkie znaki, które nie są zawarte w grupie następującej po znaku `^`. Oto kilka przykładów wyrażeń regularnych:

Wyrażenie	Znaczenie
`^[ABC]`	Znak `A`, `B` lub `C`, znajdujący się na początku ciągu znaków
`^[^ABC]`	Dowolny znak na początku ciągu znaków różny od `A`, `B` i `C`
`[^ABC]`	Dowolny znak różny od `A`, `B` i `C`
`A$`	Znak `A` znajdujący się na końcu ciągu znaków

`^A$`	Ciąg znaków, którego jedynym elementem jest `A`
`^[^a-z]$`	Ciąg znaków składający się z dokładnie jednego znaku innego niż mała litera
`^..$`	Dowolny ciąg składający się z dokładnie dwóch znaków
`\.$`	Ciąg, w którym kropka jest ostatnim elementem ciągu

Wszystkie znaki występujące wewnątrz klasy znaków reprezentują same siebie, oprócz znaku `\`, znaku `^` na początku i znaku − znajdującego się pomiędzy dwoma znakami. Na przykład, `[.]` odpowiada kropce, a notacja `^[^^]` oznacza dowolny znak różny od `^`, znajdujący się na początku łańcucha znaków.

W celu łączenia wyrażeń regularnych wykorzystywane są następujące operatory:

Operatory	**Znaczenie**
`A | B`	Alternatywa, oznacza `A` lub `B`
`AB`	Łączność, oznacza `A`, po którym następuje `B`
`A*`	Dopełnienie; odpowiada zerowej lub większej od zera liczbie znaków `A`
`A+`	Dopełnienie dodatnie; `A+` odpowiada jednemu znakowi `A` lub większej ich liczbie
`A?`	Zero lub jeden znak; odpowiada pustemu ciągowi znaków lub jednemu znakowi `A`
nawiasy	Zapis `(`*`w`*`)` odpowiada wyrażeniu regularnemu *w*

Nawiasy `()` służą grupowaniu części składowych. Są dwa operatory, alternatywy | i łączenia. Alternatywa *w1* | *w2* odpowiada dowolnemu łańcuchowi określonemu wyrażeniem `w1` lub `w2`.

Jeżeli `w1` i `w2` są wyrażeniami regularnymi, to `(`*w1*`)(`*w2*`)` (uwaga: między `(`*w1*`)` i `(`*w2*`)` nie ma spacji) odpowiada ciągowi postaci *xy*, gdzie *w1* odpowiada *x*, a *w2* odpowiada *y*. Nawiasy wokół *w1* i *w2* można pominąć, jeśli w wyrażeniach regularnych, które się w nich znajdują, nie ma operatora alternatywy.

Symbole `*`, + i `?` są operatorami stosowanymi do zapisu powtórzeń w wyrażeniach regularnych.

Przykłady.

Wyrażenie	**Znaczenie**
`B*`	Dowolny ciąg pusty
`AB*C`	`AC`, `ABC` lub `ABBC` itd. (zero, jedno lub wiele powtórzeń znaku `B`)
`AB+C`	`ABC`, `ABBC` lub `ABBBC` itd. (jedno lub wiele powtórzeń znaku `B`)
`AB?C`	`AC` lub `ABC`
`[A-Z]+`	Łańcuch wielkich liter, składający się z co najmniej jednego znaku
`[A-Z][A-Z]*`	jw.
`[A-Z]*`	Brak wielkich liter lub dowolna ich liczba (czyli dowolny ciąg znaków)
`(AB)+C`	`ABC`, `ABABC` lub `ABABABC` itd.

Kolejność operatorów według zasady pierwszeństwa (od najniższego priorytetu) jest następująca: alternatywa `|`, łączenie, operatory powtórzeń `*`, + i `?`. Podobnie, jak w wyrażeniach arytmetycznych, najpierw uwzględniane są operatory o wyższym priorytecie.

Wynika stąd, że często można pominąć nawiasy. Wyrażenie `ab | cd` jest równoważne wyrażeniu `(ab) | (cd)`, a `^ab | cd*e$` znaczy to samo, co `(^ab) | (c(d*))e$`.

Filtry `grep`

Filtry grep (`grep`, `egrep` i `fgrep`) wywodzą się z edytora `ed`, w którym istnieje polecenie `g/re/p`, gdzie `re` oznacza wyrażenie regularne. Nazwa jest skrótem słów Global Regular Expression Print. Oto ogólna postać poleceń `grep`:

```
$ grep [opcje] wzorzec [pliki]
$ grep [opcje] [-e wzorzec | -f plik] [pliki]
```

`grep` wyszukuje w *`plikach`* (lub w tekście przekazanym na standardowym wejściu) wiersze zawierające *`wzorzec`*. Domyślną czynnością jest drukowanie znalezionych wierszy. Polecenie `egrep` jest równoważne `grep -E`, a `fgrep` odpowiada `grep -F`. W następującej tabeli przedstawione są opcje tego programu i ich znaczenie.

Opcja	**Znaczenie**
`-A` *`n`* lub `-after-context=`*`n`*	Drukowanie *`n`* kolejnych wierszy po znalezieniu wiersza pasującego do wzorca; grupy wierszy oddzielone są znakami `--`
`-a` lub `--text`	Przetwarzanie pliku binarnego tak, jak pliku tekstowego; opcja równoważna `--binary-files=`*`typ`
`-B` *`n`* lub `--before-context=`*`n`*	Drukowanie *`n`* wierszy poprzedzających dopasowane wiersze; grupy wierszy oddzielone są znakami `--`
`-C` *`n`* lub `--context=`*`n`*	Drukowanie *`n`* wierszy przed dopasowanym wierszem i po nim; grupy wierszy oddzielone są znakami `--`
`-b` lub `--byte-offset`	Drukowanie liczby bajtów poprzedzających wiersz pasujący do wzorca, licząc od początku pliku
`--binary-files=`*`typ`*	Jeżeli początkowe bajty pliku wskazują na to, że plik jest plikiem binarnym, przetwarzany jest jako plik typu *`typ`*. Domyślnie *`typ`* ma wartość `binary`. W przypadku wykonywania polecenia `grep` na pliku binarnym drukowany jest krótki komunikat `binary file matches`, jeśli wzorzec pasuje do jakiegoś ciągu znaków w pliku. W przeciwnym razie nie ma żadnego komunikatu. Jeżeli *`typ`* ma wartość `without-match`, grep przyjmuje, że w pliku binarnym nie ma żadnych ciągów pasujących do wzorca, co jest równoważne opcji `-I`. Jeżeli typ ma wartość text, plik binarny przetwarzany jest tak, jak plik tekstowy, co jest równoważne opcji `-a`. Uwaga: użycie opcji `-binary-files=text` może w niektórych przypadkach dawać nieprzewidziane konsekwencje, jeśli wyjście polecenia jest kierowane na terminal, ponieważ program zarządzający pracą terminala może interpretować niektóre z otrzymanych ciągów jako polecenia
`--colour[=`*`kiedy`*`]` lub `-color[=`*`kiedy`*`]`	Wyróżnienie dopasowanego ciągu w sposób określony wartością zmiennej środowiskowej `GREP_COLOR`; *`kiedy`* przyjmuje wartości `never`, `always` lub `auto`
`-c` lub	Drukowanie liczby dopasowanych wierszy w każdym z

`--count`	plików wejściowych. Dopasowane wiersze nie są drukowane
`-D` *`czynność`*	Jeżeli plikiem wejściowym jest urządzenie, kolejka FIFO lub gniazdo, strumień danych jest przetwarzany zgodnie z dyrektywą *`czynność`*. Domyślnie *`czynność`* ma wartość `read`, co oznacza, że dane otrzymywane z urządzeń są przetwarzane tak, jakby pochodziły z pliku. Jeżeli *`czynność`* ma wartość `skip`, dane pochodzące z urządzenia nie są przetwarzane
`-d` *`czynność`* lub `--directories=`*`czynność`*	Jeżeli plikiem wejściowym jest katalog, przetwarzanie odbywa się zgodnie z dyrektywą *`czynność`*. Domyślnie czynność ma wartość `read`, co oznacza, że katalogi są odczytywane tak, jak zwykłe pliki. Jeżeli wartością jest `skip`, katalogi są pomijane bez wyświetlania żadnego komunikatu. Jeżeli czynność ma wartość `recurse`, rekursywanie odczytywane są wszystkie pliki ze wszystkich podkatalogów danego katalogu; jest to równoważne opcji `-r`
`-E` lub `--extended-regexp`	Interpretowanie wzorca jako rozszerzonego wyrażenia regularnego
`-e` *`wzorzec`* lub `--regexp=`*`wzorzec`*	Ten sposób deklaracji wzorca umożliwia przetwarzanie wzorców rozpoczynających się znakiem -
`-F` lub `--fixed-strings`	Traktowanie wzorca jako listy stałych ciągów znaków oddzielonych od siebie znakiem nowego wiersza. Dopasowywany jest każdy z tych ciągów. Użycie dodatkowej opcji `-P` lub `--perl-regexp` powoduje, że wzorzec jest traktowany jako wyrażenie regularne języka Perl
`-f` *`plik`* lub `--file=`*`plik`*	Wzorce pobierane są z pliku, po jednym z każdego wiersza. Plik pusty zawiera zero wzorców i w związku z tym nie pasuje do żadnego tekstu
`-G` lub `--basic-regexp`	Wzorzec jest interpretowany jako podstawowe wyrażenie regularne. W podstawowych wyrażeniach regularnych metaznaki `? + { | ( )` tracą specjalne znaczenie. Aby je przywrócić, należy użyć sekwencji `\? \+ \{ | \(` i `\)`
`-H` lub `--with-filename`	Drukuje nazwę pliku dla każdego dopasowanego wiersza
`-h` lub `--no-filename`	W przypadku przeszukiwania wielu plików nie drukuje nazwy pliku po znalezieniu pasującego wiersza
`--help`	Drukowanie krótkiego objaśnienia
`-I`	Przetwarzanie pliku binarnego tak, jak gdyby nie zawierał ciągów pasujących do wzorca; równoważne opcji `--binary-files=without-match`
`-i` lub `--ignore-case`	Ignorowanie wielkości liter zarówno we wzorcu, jak i w przetwarzanych plikach
`-L` lub `--files-without-match`	Dopasowane wiersze nie są drukowane. Drukowane są nazwy plików, w których nie znaleziono żadnych pasujących wierszy. Przetwarzanie pliku jest zakończone

	po znalezieniu pierwszego dopasowania
`-l` lub `--files-with-matches`	Dopasowane wiersze nie są drukowane. Drukowane są nazwy plików, w których znaleziono pasujący wiersz. Przetwarzanie pliku jest zakończone po znalezieniu pierwszego dopasowania
`-m` *`n`* lub `--max-count=`*`n`*	Zakończenie wczytywania pliku po znalezieniu *`n`* pasujących wierszy. W przypadku użycia opcji `-v` lub `--invert-match` przetwarzanie jest zakończone po wydrukowaniu *`n`* wierszy nie zawierających wzorca
`--mmap`	Jeśli to możliwe, wczytywanie danych odbywa się za pomocą funkcji systemowej `mmap(2)` zamiast domyślnej funkcji `read(2)`. W niektórych sytuacjach daje to większą wydajność. Związane jest z tym jednak pewne ryzyko (mogą być wykonane zrzuty stanu pamięci), gdy plik wejściowy się zmniejszy w czasie działania programu grep lub pojawi się błąd operacji I/O
`-n` lub `--line-number`	Na początku każdego wiersza drukowany jest jego numer w pliku wejściowym
`-o` lub `--only-matching`	Drukowanie jedynie ciągów znaków pasujących do wzorca, a nie całych wierszy
`--label=`*`nazwa`*	Wyświetlanie danych pochodzących ze standardowego wejścia w taki sposób, jakby pochodziły z pliku nazwa. Jest to przydatne w przypadku korzystania z narzędzi w rodzaju zgrep, na przykład `gzip -cd` *`nazwa`*`.gz | grep --label=`*`nazwa wzorzec`*
`--line-buffering`	Buforowanie wierszy. Może skutkować obniżeniem wydajności
`-q` lub `--quiet` lub `--silent`	Na wyjściu nie są drukowane żadne wyniki. Po znalezieniu pasujących wierszy przetwarzanie jest zakończone z kodem zero, niezależnie od pojawienia się błędów. Zobacz opcja `-s` lub `--no-messages`
`-R` lub `-r` lub `--recursive`	Rekursywne przetwarzanie wszystkich plików z każdego katalogu i podkatalogów; równoważne opcji `-d`
`--include=`*`wzorzec`*	Rekursywne przetwarzanie katalogów z uwzględnieniem nazw plików pasujących do *`wzorca`*
`--exclude=`*`wzorzec`*	Rekursywne przetwarzanie katalogów z pominięciem nazw plików pasujących do *`wzorca`*
`-s` lub `--no-messages`	Pominięcie komunikatów o nie istniejących plikach lub problemach z odczytaniem plików
`-U` lub `-binary`	Pliki są traktowane jako pliki binarne
`-u` lub `--unix-byte-offsets`	Drukowanie liczby bajtów poprzedzających wiersz pasujący do wzorca, licząc od początku pliku; zakłada się format plików tekstowych Uniksa – znaki `CR` są pominięte
`-V` lub `--version`	Drukowanie numeru wersji programu grep na standardowym wyjściu błędów
`-v` lub `--invert-match`	Zanegowanie dopasowania; drukowane są jedynie wiersze nie pasujące do wzorca

`-w` lub `--word-regexp`	Drukowane są jedynie wiersze, w których dopasowane wyrażenia tworzą pełne słowa. Pasujący ciąg znaków musi znajdować się na początku wiersza lub być poprzedzony znakiem nie tworzącym słów. Podobnie za słowem musi znajdować się koniec wiersza lub znak nie tworzący słów. Do znaków tworzących słowa zaliczają się litery, cyfry i podkreślniki
`-x` lub `--line-regexp`	Drukowanie jedynie wierszy w całości zgodnych z wzorcem (oprócz ciągu znaków zgodnego z wzorcem nie zawierają żadnych innych znaków)
`-y`	Synonim opcji `-i`
`-Z` lub `--null`	Drukowanie znaku `NUL` (kod ASCII 000) zamiast znaku, który zwykle pojawia się po nazwie pliku. Na przykład, `grep -lZ` drukuje `NUL` po każdej nazwie pliku zamiast znaku nowego wiersza. Ułatwia to przetwarzanie nazw plików zawierających nietypowe znaki, na przykład znak nowego wiersza

W następującej tabeli zamieszczone są przykłady stosowania filtrów `grep`.

Polecenie	**Znaczenie**
`fgrep abc` *`plik1`*	Drukowanie wierszy pliku *`plik1`* zawierających ciąg `abc`
`fgrep -f` *`plik1 plik2`*	Drukowanie wierszy z pliku *`plik1`* zawierających wyrażenia zapisane w *`plik2`*
`grep abc` *`plik`*	Drukowanie wierszy zawierających ciąg `abc`
`grep '^abc'` *`plik`*	Drukowanie wierszy rozpoczynających się ciągiem `abc`
`grep '^a'` *`plik`*	Drukowanie wierszy rozpoczynających się literą `a`
`grep '^[a-z]'` *`plik`*	Drukowanie wierszy rozpoczynających się małą literą
`grep '^.[ab]'` *`plik`*	Drukowanie wierszy, w których drugą literą jest `a` lub `b`
`grep '\.$'` *`plik`*	Drukowanie wierszy, w których ostatnim znakiem jest `.` (kropka)
`grep ';$'` *`plik`*	Drukowanie wierszy kończących się znakiem `;` (średnik)
`grep -c ';'`	Liczenie wierszy kodu programu w języku C
`grep '\.\.*'` *`plik`*	Drukowanie wierszy kończących się co najmniej jedną kropką
`egrep ';+'` *`plik`*	Drukowanie wierszy zawierających co najmniej jeden średnik
`egrep 'a(_)?1'` *`plik`*	Drukowanie wierszy zawierających ciąg `a1` lub `a_1`
`egrep '(a1)|(a_1)'` *`plik`*	Drukuje wiersze zawierające `a1` lub `a_1`
`egrep -f` *`plik1 plik2`*	Drukuje te wiersze pliku *`plik1`*, które zawierają wyrażenia zapisane w *`plik2`*

Oto prosty przykład użycia polecenia `grep`. Plik `znajomi` ma następującą zawartość:

```
Ola             Poznan    61 6119347
Jacek           Warszawa  22 9203205
Ania            Krakow    12 4312580
```

```
Piotr           Gdansk    58 7639209
```

Trzecie i czwarte pole w tym pliku to numer kierunkowy i numer telefonu stacjonarnego. Aby wyświetlić numer telefonu Jacka, należy wykonać następujące polecenie:

```
$ grep 'Jacek' znajomi
```

Wynik jest następujący:

```
Jacek           Warszawa  22 9203205
```

Przykłady

Sprawdzenie, czy w systemie zalogowany jest użytkownik `nowak`:

```
$ who | grep nowak
```

Sprawdzenie daty logowania użytkownika do systemu:

```
$ last | grep nowak
```

Jeżeli logowań było wiele i nie mieszczą się na jednym ekranie, bardziej przydatne jest następujące polecenie:

```
$ last | grep nowak | less
```

Przeszukiwanie wszystkich plików bieżącego katalogu w poszukiwaniu słowa `Piotr`. Drukowane są nazwy plików i zawartość dopasowanych wierszy:

```
$ grep 'Piotr' *
znajomi: Piotr      Gdansk    58 7639209
```

Wyświetlenie nazw katalogów znajdujących się w bieżącym katalogu:

```
$ ls -l | grep '^d'
```

Strumieniowy edytor tekstu `sed`

Nazwa `sed` jest skrótem słów Stream EDitor. Narzędzie to powstało w 1973 lub 1974 r. Jego autorem jest Lee E. McMahon. Edytor `sed` działa w trybie nieinterakcyjnym (wsadowym). Przetwarza kolejne wiersze wejściowego strumienia tekstu, dokonuje w nich zmian zapisanych w instrukcjach i przekazuje wynik do standardowego wyjścia. Wykorzystywane są dwa bufory danych: przestrzeń wzorca (ang. *pattern space*) i przestrzeń tymczasowa (ang. *hold space*). Pojedynczy wiersz po wczytaniu przez `sed` trafia do przestrzeni wzorca, w której wykonywane są manipulacje tekstu. Przestrzeń tymczasowa jest początkowo pusta, ale za pomocą niektórych poleceń można przekazywać do niej dane z przestrzeni wzorca i w odwrotnym kierunku.

Oto ogólna postać polecenia `sed`:

`sed [`*`opcje`*`] `*`polecenie_edycji pliki`*

Polecenie edycji ma następującą postać:

`[`*`adres1`*`[,`*`adres2`*`]][`*`funkcja`*`][`*`argumenty`*`]`

Adresy nie są obowiązkowe i mogą być oddzielone od funkcji spacją lub tabulatorem.

Rolę adresów wskazujących konkretne wiersze tekstu mogą pełnić liczby dziesiętne. Pierwszy przetwarzany wiersz jest oznaczony numerem 1. W przypadku przetwarzania wielu plików tekst wszystkich plików jest traktowany jak jedna całość i adresy wierszy są stale zwiększane. Ostatni przetwarzany wiersz jest oznaczany znakiem `$`. Oprócz adresów liczbowych używane są też adresy kontekstowe, którymi są wzorce z wyrażeniami regularnymi ograniczone ukośnikami `/`.

Polecenia edycji mogą zawierać 0, 1 lub 2 adresy oddzielone przecinkami. Zakres działania poleceń edycji jest wówczas następujący:

Liczba adresów	**Zakres działania**
0	Wszystkie wiersze tekstu
1	Wiersz o wskazanym adresie
2	Wszystkie wiersze w zakresie od wiersza wskazanego przez pierwszy adres do wiersza odpowiadającego drugiemu adresowi. Czynności edycji są następnie powtarzane w następnym zakresie wierszy, do którego pasują podane adresy.

Polecenia powinny być umieszczane w pojedynczym cudzysłowie `'`, jeśli określone są dodatkowe opcje lub funkcje. Wzorce pełnią tę samą rolę, co adresy kontekstowe. Różnią się od adresów kontekstowych tym, że ich ogranicznikami mogą być nie tylko spacja i znak otwarcia nowego wiersza, ale także wszystkie zwykłe znaki tekstowe. Wstawiany łańcuch nie jest wyrażeniem regularnym. Metaznak poprzedzony ukośnikiem `\` jest pozbawiony specjalnego znaczenia.

W tabeli są umieszczone najczęściej używane opcje edytora `sed`.

Opcja	**Znaczenie**
`-e`	Traktuj następny argument jako instrukcję edytora
`-n`	Nie drukuj domyślnego wyjścia, a jedynie wiersze wybrane instrukcjami `p` lub `s///p`
`-f` *skrypt*	Pobierz skrypty edycji z pliku *skrypt*

Do najczęściej używanych instrukcji `sed` należy instrukcja podstawiania:

```
s/wzorzec/nowy_łańcuch/znacznik
```

W miejsce wszystkich ciągów pasujących do *wzorca* podstawiany jest *nowy_łańcuch*. Wzorzec jest wyrażeniem regularnym. Przetworzony tekst pojawia się na standardowym wyjściu, czyli na ekranie. Wyjście można skierować do innego pliku:

```
$ sed [opcje] 'polecenia' plik > plik1
```

Polecenia edycji mogą być zapisane w pliku. Wywołuje się je za pomocą opcji `-f`:

```
$ sed -f plik_poleceń plik > plik1
```

Powiedzmy, że w pliku `miasta` znajduje się następujący tekst:

```
Warszawa
Katowice
Berlin
Kopenhaga
Londyn
Madryt
```

Aby umieścić w pliku informację o kraju, w którym znajduje się miasto, można wykonać następujący skrypt umieszczony w pliku `dodajkraj.sed`:

```
s/Warszawa/Warszawa PL/p
s/Katowice/Katowice PL/p
s/Berlin/Berlin DE/p
s/Kopenhaga/Kopenhaga DK/p
s/Londyn/Londyn GB/p
s/Madryt/Madryt ES/p
```

Odpowiednie polecenie ma następującą postać:

```
$ sed -f dodajkraj.sed miasta > miasta1
```

Domyślnie `sed` drukuje cały plik, zarówno wiersze, które uległy zmianie, jak i te, które pozostały bez zmian. Aby drukować jedynie część pliku, należy użyć opcji `-n`, która blokuje standardowe wyjście (żaden tekst nie pojawia się na wyjściu). Opcja `-n` powoduje, że drukowane są jedynie wiersze wybrane poleceniem `p` lub znacznikiem `p` polecenia `s`. Aby wydrukować zawartość pliku `miasta1` od wiersza zawierającego `Berlin` do wiersza zawierającego `Londyn`, należy wykonać następujące polecenie:

```
$ sed –n '/Berlin/,/Londyn/p' miasta1
Berlin DE
Kopenhaga DK
Londyn GB
```

Opcja `-e` informuje, że następny argument jest poleceniem edycji. Jest to potrzebne przy dwóch poleceniach lub większej ich liczbie, na przykład:

```
$ sed –e 's/ PL/, Polska/' –e 's/ DE/, Niemcy/' miasta1
Warszawa, Polska
Katowice, Polska
Berlin, Niemcy
Kopenhaga DK
Londyn GB
Madryt ES
```

Wiersze, w których należy wykonać podstawienie, można wybrać za pomocą wzorca:

```
$ sed '/Warszawa/s/ PL/, Polska/' miasta1
```

W tym przykładzie podstawienie wykonywane jest tylko w wierszu zawierającym słowo `Warszawa`.

Komentarze w skryptach `sed` są oznaczane znakiem `#`. Wiersz jest traktowany jako komentarz, jeżeli `#` jest pierwszym znakiem wiersza.

W łańcuchu, który jest wstawiany w miejsce wyrażenia regularnego (wzorca) jedynie następujące znaki mają znaczenie specjalne:

Znak	**Znaczenie**
`&`	W to miejsce wstawiany jest łańcuch pasujący do wyrażenia regularnego
`\n`	Pasuje do *n*-tego podciągu zdefiniowanego we wzorcu za pomocą znaków `\(` i `\)`
`\`	Używany do zmiany specjalnego znaczenia znaków `&`, `\` i ogranicznika polecenia podstawienia tekstu

Następująca tabela zawiera polecenia edytora `sed`:

Polecenie	**Znaczenie**
`#`	Początek komentarza; musi być pierwszym znakiem wiersza
`:`*`etykieta`*	Etykieta dla poleceń `b` i `t`
`=`	Drukowanie numeru bieżącego wiersza
`a\` *`tekst`*	Dołączenie *`tekstu`*. Jeżeli dołączany *`tekst`* składa się z wielu wierszy, znaki nowego wiersza muszą być poprzedzone znakiem `\`
`i\` *`tekst`*	Wstawianie *`tekstu`*. Jeżeli dołączany *`tekst`* składa się z wielu wierszy, znaki nowego wiersza muszą być poprzedzone znakiem `\`
`r` *`plik`*	Wczytanie i dołączenie tekstu z *`pliku`*
`b` *`etykieta`*	Przetwarzanie bieżącego wiersza ma być kontynuowane w poleceniu znajdującym się po *`etykiecie`*. Jeżeli brak *`etykiety`*, przejdź na koniec skryptu

`c\`
`tekst`
W miejsce wierszy określonych adresem wstawiany jest *`tekst`*. Jeżeli dołączany *`tekst`* składa się z wielu wierszy, znaki nowego wiersza muszą być poprzedzone znakiem `\`

`d`
Usunięcie wierszy określonych adresem

`D`
Usunięcie pierwszego wiersza z przestrzeni wzorca, od początku do zamaskowanego znaku nowego wiersza. Przetwarzanie jest następnie kontynuowane od początku skryptu. Jeżeli w przestrzeni wzorca nie ma więcej wierszy, wczytywany jest nowy wiersz wejściowy

`g`
Usunięcie zawartości przestrzeni wzorca i skopiowanie do niej zawartości przestrzeni tymczasowej

`G`
Dołączenie zawartości przestrzeni tymczasowej do przestrzeni wzorca; dołączony tekst jest oddzielony znakiem nowego wiersza

`h`
Usunięcie przestrzeni tymczasowej i skopiowanie do niej zawartości przestrzeni wzorca

`H`
Dołączenie zawartości przestrzeni wzorca do przestrzeni tymczasowej; dołączony tekst jest oddzielony znakiem nowego wiersza (również wtedy, kiedy przestrzeń tymczasowa jest pusta)

`i\`
`tekst`
Wstawienie *`tekstu`* przed każdym wierszem zgodnym z adresem

`l`
Wypisanie zawartości przestrzeni wzorca; w miejsce znaków niedrukowalnych wyświetlane są kody ASCII

`n`
Wyświetlenie przestrzeni wzorca, usunięcie bieżącej zawartości i wczytanie następnego wiersza do przestrzeni wzorca; następną wykonywaną instrukcją jest kolejne po `n` polecenie skryptu

`N`
Dołączenie następnego wiersza wejściowego do przestrzeni wzorca; dołączony tekst jest oddzielony znakiem nowego wiersza

`p`
Wyświetlenie zawartości przestrzeni wzorca

`P`
Wyświetlenie pierwszego wiersza przestrzeni wzorca

`q`
Zakończenie działania po napotkaniu wiersza o określonym adresie

`r` *`plik`*
Wczytanie pliku do przestrzeni wzorca

`s`
`s/`*`wzorzec`*`/`*`nowy_łańcuch`*`/[`*`znaczniki`*`]`
Podstawienie w miejsce *`wzorca`* nowego tekstu. Można użyć następujących znaczników:

- `g` — Wykonanie podstawienia dla wszystkich wystąpień wzorca w wierszu
- *`n`* — Liczba od 1 do 512; podstawienie jest wykonywane tylko dla *`n`*-tego wystąpienia *`wzorca`* w wierszu
- `p` — Jeżeli podstawienie zostało wykonane, wiersz jest drukowany na standardowym wyjściu
- `w` *`plik`* — Jeżeli podstawienie zostało wykonane, wiersz jest wpisywany do pliku

`t`
Jeżeli polecenie `s///` zostało wykonane, przejdź do polecenia oznaczonego *`etykietą`*. W przypadku braku *`etykiety`* przejdź na koniec skryptu

`w` *`plik`*
Zapisanie zawartości przestrzeni wzorca w *`pliku`*

`x`
Zamiana zawartości przestrzeni wzorca i przestrzeni tymczasowej

`y/`*`znaki1`*`/`*`znaki2`*`/`
Translacja znaków z przestrzeni wzorca; na przykład `y/abc/123/` oznacza zmianę wszystkich wystąpień `a` na `1`, `b` na

2, c na 3

Oto przykłady poleceń `sed`.

Polecenie	**Znaczenie**
`sed s/,/,\ /g` *plik*	W miejsce każdego przecinka wstawia przecinek i spację
`sed 's/abc/def/'` *plik*	Zamienia ciąg `abc` na `def`. Zamiana dotyczy jedynie pierwszego ciągu `abc` w wierszu. Jeżeli w tym samym wierszu `abc` pojawia się więcej razy, kolejne wystąpienia `abc` nie będą zamieniane
`sed '1,4s/abc/def/'` *plik*	Zamienia `abc` na `def` w wierszach od 1 do 4. Zamiana dotyczy jedynie pierwszego ciągu `abc` w wierszu
`sed 's/abc/def/g'` *plik1 plik2*	Zamienia `abc` na `def` w plikach *plik1* i *plik2*. Znacznik `g` oznacza zamianę wszystkich wystąpień `abc` w całym wierszu
`sed 's/abc/def/gw` *zmiany*`'` *plik*	Zamienia `abc` na `def`. Zmienione wiersze są dopisywane do pliku *zmiany*
`sed 's/abc/def/g'` *plik1* `>>` *plik2*	Zamienia `abc` na `def` w całym pliku. Dopisuje zmiany do *plik2*
`sed 's/abc/def/3'` *plik*	Zamienia trzeci ciąg `abc` w każdym wierszu
`sed '10!s/pecet/notebook/'` *plik*	Zamienia `pecet` na `notebook` wszędzie, oprócz 10 wiersza
`sed 'd'` *plik*	Usuwa wszystkie wiersze
`sed '1d'` *plik*	Usuwa pierwszy wiersz
`sed '$d'` *plik*	Usuwa ostatni wiersz
`sed '/^$/d'` *plik*	Usuwa puste wiersze
`sed '12,28d'` *plik1*	Usuwa z pliku *plik1* wiersze od 12 do 28
`sed '10,20d'` *plik1* `>>` *plik2*	Usuwa z pliku *plik1* wiersze od 10 do 20, a zmienione wiersze są dopisywane do pliku *plik2*
`sed '/abc/,/def/d'` *plik*	Usuwa wszystkie wiersze od pierwszego, w którym pojawia się ciąg `abc`, do najbliższego, w którym pojawia się `def`
`sed '/\{/,/\}/d'` *plik*	Usuwa wiersze od pierwszego, w którym pojawia się znak `{`, do najbliższego, w którym pojawia się `}`
`sed '1,/abc/d'` *plik*	Usuwa wiersze od pierwszego do najbliższego, zawierającego ciąg `abc`
`sed '1,/^$/d'` *plik*	Usuwa wiersze od początku pliku do pierwszego pustego wiersza; przydatne do usuwania nagłówków poczty
`sed '1,/^$/!d'` *plik*	Zaprzeczenie poprzedniego polecenia;

	pozostawia jedynie wiersze od początku do pierwszego pustego, a pozostałe usuwa
`sed '100,$d' plik`	Usuwa wiersze od 50 do końca pliku
`sed '/\./,$d' plik`	Usuwa wiersze od pierwszego wystąpienia znaku . (kropka) do ostatniego wiersza
`sed 's/ *$//d' plik`	Usuwa wszystkie spacje na końcu każdego wiersza
`sed 's/^[ ^t]*//' plik`	Usuwa wszystkie spacje i tabulatory na początku każdego wiersza
`sed 's/[ ^t]*$//' plik`	Usuwa wszystkie spacje i tabulatory na końcu każdego wiersza
`sed 's/^[ ^t]*$//;s/[ ^]*$//' plik`	Usuwa wszystkie spacje i tabulatory na początku i na końcu każdego wiersza
`sed -n '/^Date:/,/^Name:/p' plik`	Po napotkaniu ciągu `Date:` na początku wiersza drukuje ten wiersz i wszystkie kolejne, aż do pojawienia się na początku wiersza ciągu `Name:`
`sed -n 1,10p plik`	Drukuje tylko pierwszych 10 wierszy tekstu
`sed G plik > plik1`	Między wierszami tekstu umieszcza dodatkowy pusty wiersz
`sed 'G;G' plik > plik1`	Między wierszami tekstu umieszcza dwa puste wiersze
`sed -n '/[0-9]\{3\}/p' plik`	Drukuje wiersze zawierające ciąg 3 cyfr

Jak szybko sprawdzić, czy zmiany wprowadzone w pliku tekstowym są zgodne z oczekiwaniami? Po wykonaniu polecenia

```
$ sed -f skrypt plik > plik1
```

należy porównać zawartość obu plików:

```
$ diff plik plik1
```

Do jednego adresu lub grupy adresów można podać grupę poleceń. W takim przypadku należy wszystkie polecenia umieścić w nawiasie klamrowym:

```
$ sed '/ticket/{s/bus/train/g;s/cash/credit card/g;}' plik
```

Zapis jest bardziej przejrzysty, jeśli umieścimy poszczególne instrukcje w oddzielnych wierszach:

```
$ sed '/ticket/{
s/bus/train/g
s/cash/credit card/g
}' plik
```

Podczas przetwarzania tekstu dość często potrzebne jest umieszczenie poszczególnych słów (ciągów znaków) w osobnych wierszach. Oto polecenie realizujące ten zamiar:

```
$ sed 's/[ ][ ]*/\
>/g' plik
```

Po napisaniu pierwszego wiersza należy nacisnąć klawisz **Enter**. Znak > pojawiający się na początku drugiego wiersza jest znakiem kontynuacji polecenia drukowanym przez powłokę. Nawiasy klamrowe są oddzielone od siebie spacją. Jeżeli między nawiasami nie ma spacji, w każdym wierszu znajdzie się jedna litera tekstu.

```
$ sed –n '/abc/w plik1
          /abc/!w plik2' plik
```

Wyrażenia regularne, będące argumentami poleceń, mogą być ograniczane innymi znakami niż ukośnik. Jeżeli w tych wyrażeniach zawarte są ukośniki, polecenie można zapisać na przykład za pomocą wykrzykników:

```
  s!/usr/bin!/usr/local/bin!
```

lub w następujący sposób:

```
  s|/usr/bin|/usr/local/bin|
```

Jeżeli znak ograniczający wyrażenie regularne pojawia się w samym wyrażeniu lub w zmienianym tekście, należy poprzedzić go znakiem `\`, aby zmienić jego znaczenie z metaznaku na zwykły znak.

Oto kilka następnych przykładów. Usunięcie pustych wierszu na początku pliku:

```
$ sed '/./,$!d' plik
```

Usunięcie pustych wierszy na końcu pliku:

```
$ sed –e :a –e '/^\n*$/{$d;N;};/\n$\ba
```

Usunięcie wierszy komentarza, ograniczonego znakami `/*` i `*/`, z programu w języku C:

```
$ sed –e '/^\/\*/d' –e '/.*\/\*/d' program.c
```

Jeżeli wiersz kończy się znakiem `\`, dołącz do niego następny wiersz:

```
$ sed –e :a –e '/\\$/N; s/\\\n//; ta' plik
```

Jeżeli wiersz rozpoczyna się znakiem =, dołącz go do poprzedniego wiersza, zamieniając przy tym znak = na spację:

```
$ sed –e :a –e '$!N;s/\n=/ /;ta' –e 'P;D' plik
```

Jak wykonać polecenie `sed` na wszystkich plikach w bieżącym katalogu? Oto przykład, w którym ciąg `def` zastępuje ciąg `abc` we wszystkich plikach z rozszerzeniem `.txt`:

```
#!/bin/sh
# Plik nazwa.txt.tmp zawiera poprzednią wersję pliku.
# Dzięki temu można anulować zmiany w przypadku
# pojawienia się błędów.
for plik in *.txt
do
  cp $plik $plik.tmp
  sed 's/abc/def/g' $plik.tmp > $plik
done
```

W celu wykonania polecenia edytora w całym drzewie katalogowym można użyć instrukcji `find`:

```
#!/bin/sh
find . -type f -name '*.txt' -print | while read i
do
  sed 's/abc/def/g' $i > $i.tmp && mv $i.tmp $i
done
```

Instrukcję można zapisać też w innej postaci, na przykład:

```
  sed 's|abc|def|g' $i > $i.tmp && mv $i.tmp $i
```

Ma to znaczenie w przypadku edycji łańcuchów zawierających znak `/` – wówczas ukośnik nie może pełnić roli ogranicznika. Skrypt można nieco uogólnić, używając zmiennych domyślnych powłoki:

```
  sed "s|$1|$2|g" $i > $i.tmp
```

Tym razem niezbędny jest podwójny cudzysłów, aby zmienne `$1` i `$2` mogły być interpretowane przez powłokę. Zmienne `$1` i `$2` można przekazać do skryptu jako argumenty (zobacz rozdział *Skrypty powłokowe*):

```
$ ./skrypt abc def
```

Podobnie można postąpić w przypadku zmiennych środowiskowych:

```
  sed "s/$user"'/root/'
```

Oto dwie wersje skryptu, w którym wielkie litery są zamieniane na małe:

```
#!/bin/sh
# Zamiana wielkich liter na małe
sed 'y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/abcdefghijklmnopqrstuvwxyz/
' $1 > $2
```

```
#!/bin/sh
# Zamiana wielkich liter na małe
DUZE='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
male='abcdefghijklmnopqrstuvwxyz'
```

```
sed 'y/'"$DUZE"'/'"$male"'/' $1 > $2
```

Skrypt jest uruchamiany za pomocą następującego polecenia:

```
$ ./skrypt plik1 plik2
```

Wynik jest zapisany w *pliku2*.

W następnym przykładzie usuwane są spacje lub tabulatory na początku i na końcu wiersza. Pozostałe ciągi spacji i tabulatorów są zastąpione pojedynczymi spacjami.

```
#!/bin/sh
sed 's/^[ ]*//;  s/[ ]*$//;  s/[ ][ ]*/ /g' $1
```

Następujący skrypt umieszcza poszczególne ciągi znaków w osobnych wierszach:

```
#!/bin/sh
sed '/^[ ]*$/d;  s/^[ ]*//;  s/[ ]*$//;  s/[ ][ ]*/\
/g' $1
```

Drukowanie 5-cyfrowej liczby pseudolosowej:

```
$ head –c4 /dev/urandom | od –N2 –tu4 | sed –ne '1s/.* //p'
```

Plik `/dev/urandom` generuje liczby pseudolosowe. Polecenie `od` przekształca wynik do czytelnej postaci, a `sed` odrzuca początkowe zera i spacje (przykład z rubryki Shell Corner w witrynie `www.unix.review`, May 2003).

Język awk

Awk jest językiem programowania do szeroko rozumianego przetwarzania tekstu i danych. Nazwa `awk` pochodzi od pierwszych liter nazwisk jego autorów: Alfreda V. Aho, Briana W. Kernighana i Petera J. Weinbergera. Pierwsza wersja tego narzędzia powstała w 1977 r. w AT&T Bell Laboratories. Składnia awk jest częściowo podobna do języka C. Z czasem pierwotna wersja `awk` została zastąpiona przez `nawk` (*new awk*). Free Software Foundation stworzyła własną, rozszerzoną wersję tego języka pod nazwą `gawk` (*GNU awk*). `gawk` zawiera wszystkie elementy składni `nawk`. Materiał tego rozdziału jest dotyczy wersji `nawk`.

Oto najprostsze uruchomienie interpretera `awk` w wierszu poleceń:

```
$ awk 'program' pliki
```

gdzie *program* oznacza wyrażenie postaci

```
wzorzec { instrukcje }
```

Wzorzec jest wyrażeniem wybierającym wiersze, na których należy wykonać *instrukcje*. Jest kilka rodzajów takich wyrażeń (patrz niżej). Jeśli wzorzec jest pominięty, przetwarzane są wszystkie wiersze:

```
$ awk '{ instrukcje }' pliki
```

Na przykład, polecenie

```
$ awk '{ print }' pliki
```

powoduje wydrukowanie na wyjściu całej zawartości plików. Wykonuje więc to samo zadanie, co polecenie `cat`, tyle że wolniej.

Uwaga. Między nawiasami klamrowymi, a zawartymi w nich instrukcjami, konieczne są spacje (choć niektóre wersje `awk` dopuszczają pominięcie tych spacji).

Jeżeli nie są zdefiniowane żadne czynności, drukowane są wszystkie wiersze pasujące do danego wzorca:

```
$ awk 'wzorzec' pliki
```

Awk nie zmienia postaci plików wejściowych. Wiersze plików wejściowych są przetwarzane jak nieprzerwany strumień danych. Polecenia `awk` można wczytywać z pliku:

```
$ awk -f plik_poleceń pliki
```

Każdy wiersz przetwarzany przez `awk` jest automatycznie dzielony na pola (ang. *fields*), nieprzerwane ciągi znaków, oddzielone spacjami lub tabulatorami. Jest to domyślne znaczenie spacji i tabulatorów (znaki rozdzielające pola można jednak zdefiniować dowolnie). Według tej definicji, polecenie `ls -l` wyświetla informacje w dziewięciu polach:

```
$ ls -l
-rw-r--r--    1 root     root      1973 Mar 12 01:12 inetd.conf
-rw-r--r--    1 root     root      6912 Mar 22  2002 mime.conf
-rw-r--r--    1 root     root       386 Mar 12 01:12 profile
drwxr-xr-x    1 root     root         0 Mar 12 01:06 profile.d
-rw-r--r--    1 root     root     12306 Mar 12 01:31 termcap
```

Domyślnie pola numerowane są `$1`, `$2`, ..., `$NF`, gdzie `NF` jest zmienną równą liczbie pól w danym wierszu. W tym przykładzie w każdym wierszu jest 9 pól. Należy odróżnić `NF`, czyli liczbę pól w wierszu, od `$NF` – numeru ostatniego pola w wierszu. Jeżeli wyświetlona mają być jedynie uprawnienia plików, rozmiar i nazwa, to należy wykonać następujące polecenie:

```
$ ls -l | awk '{ print $1, $5, $9 }'
-rw-r--r-- 1973 inetd.conf
-rw-r--r-- 6912 mime.conf
-rw-r--r-- 386 profile
drwxr-xr-x 0 profile.d
-rw-r--r-- 12306 termcap
```

Łatwo jest posortować pliki według ich rozmiaru:

```
$ ls -l | awk '{ print $5, $9 }' | sort -n
0 profile.d
386 profile
1973 inetd.conf
6912 mime.conf
12306 termcap
```

W `awk` domyślnie zdefiniowana jest zmienna `NR` (ang. *number of record*), która jest numerem rekordu. Domyślnie każdy wiersz jest oddzielnym rekordem, a `NR` numeruje kolejne wiersze. Oto prosty sposób na ponumerowanie wierszy pliku wejściowego:

```
$ awk '{ print NR, $0 }' plik
```

Pole `$0` oznacza całą zawartość rekordu. Format wydruku można dokładniej określić za pomocą instrukcji `printf`:

```
$ awk '{ printf "%6d %s \n", NR, $0 }'
```

Notacja `%6d` oznacza całkowitą liczbę dziesiętną zapisaną w polu o szerokości 6 znaków, `%s` oznacza ciąg znaków, a `\n` otwiera nowy wiersz.

Następny przykład ilustruje zastosowanie wzorców:

```
$ ls -l | awk '$5 > 10000 { print }'
-rw-r--r--    1 ktos     unknown     12306 Mar 12 01:31 termcap
```

Wynik polecenia `ls -l` jest przekazywany do `awk`. Drukowane są jedynie te wiersze, w których zawartość piątego pola jest większa niż 10000.

Załóżmy, że plik `znajomi` zawiera imiona i nazwiska osób oraz adresy poczty elektronicznej:

```
Maria     Kowalska       M.Kowalska@firma.pl
Grzegorz  Nowak               G.Nowak@linux.org.
Anna      Lewandowska
```

Sprawdźmy, czy są wiersze, w których liczba pól jest mniejsza niż 3:

```
$ awk 'NF < 3 { print }' znajomi
Anna Lewandowska
```

Powoduje to wyświetlenie ostatniego wiersza, zawierającego jedynie dwa pola. Tę samą informację można uzyskać za pomocą następujących warunków:

`$3 == ""`	trzecie pole jest puste
`$3 ~ /^$/`	trzecie pole jest pustym ciągiem znaków
`$3 !~ /./`	trzecie pole nie zawiera żadnego znaku
`length($3) == 0`	długość trzeciego pola jest równa 0

Funkcja `length` zwraca długość ciągu znaków. Jeśli funkcja jest wywołana bez argumentu, domyślnie jest interpretowana jako długość całego wiersza. Całość wzorca można zanegować w następujący sposób:

`!($3 == "")`	trzecie pole nie jest puste

Oto inne przykłady wzorców:

`NF %2 != 0`	liczba pól jest nieparzysta; znak `%` oznacza resztę z dzielenia liczb całkowitych
`length($0) > 30`	długość wiersza przekracza 30 znaków

Oto przykład, w którym drukowane są wiersze zawierające więcej, niż 30 znaków.

```
awk 'length($0) > 30 { print "dlugi", "wiersz:",
substr($0,1,30) }' plik
```

Po komunikacie `dlugi wiersz` drukowanych jest 30 znaków wiersza. Funkcja `substr(a,n1,n2)` wybiera z łańcucha `a` znaki od *n1* do *n2*.

Do grupy wzorców należą również dwa wyrażenia specjalne: `BEGIN` i `END`. Czynności przyporządkowane wyrażeniu `BEGIN` są wykonywane przed przeczytaniem pierwszego wiersza. Można to wykorzystać do inicjalizacji zmiennych, drukowania nagłówków lub np. do zdefiniowania separatora pól, czyli zmiennej `FS` (ang. *field separator*).

Przykład. Plik adresów poczty elektronicznej ma następującą postać:

```
Maria:Kowalska:M.Kowalska@firma.pl
Grzegorz::G.Nowak@linux.org
```

```
Anna:Lewandowska
```

Chcemy sprawdzić, czy są wiersze, w których drugie pole jest puste:

```
$ awk 'BEGIN { FS = ":" }
  $2 == "" ' plik
```

W rezultacie wydrukowany zostanie drugi wiersz. Czynności związane z wyrażeniem `END` zostaną wykonane po przetworzeniu wszystkich wierszy podanych na wejściu. Na przykład, drukuje liczbę wszystkich wierszy pliku:

```
$ awk 'END { print NR }' plik
```

Uwaga: puste wiersze też są zliczane! Przy wykonywaniu różnego rodzaju obliczeń należy o tym pamiętać, aby nie popełnić błędów. Można sobie z tym poradzić, podając we wzorcu wyrażenie odpowiadające tylko wierszom niepustym. Wzorce mogą także mieć postać zakresu:

```
wzorzec1, wzorzec2
```

Drukowane są wówczas wszystkie wiersze, od pojawienia się *wzorca1* do najbliższego wystąpienia *wzorca2*.

Prosta arytmetyka

Oprócz przetwarzania tekstu, główną zaletą `awk` jest możliwość wykonywania obliczeń. Następujący skrypt oblicza sumę liczb z pierwszej kolumny i ich średnią:

```
{ a = a + $1 }
END { print a, a/NR }
```

W celu łatwiejszego odczytania wyników drugi wiersz można zapisać w następującej postaci:

```
END {print "suma:", a, "srednia:", a/NR }
```

Na przykład, jeżeli plik zawiera następujące dane:

```
3.2
4
5.3
```

wynik skryptu jest następujący:

```
suma:  12.5  srednia:  4.16667
```

Podobnie, jak w języku C, wyrażenie `s = s + $1` można zapisać krócej: `s += $1`.

Poznaliśmy już niektóre zmienne zdefiniowane domyślnie w `awk`. Oto pełna lista (dotyczy wersji `nawk`):

Zmienna	Znaczenie
ARGC	Liczba argumentów wiersza poleceń
ARGV	Tabela z argumentami wiersza poleceń
FILENAME	Nazwa przetwarzanego pliku
FNR	Numer bieżącego rekordu w bieżącym pliku
FS	Separator pól (domyślnie spacja)
NF	Liczba pól rekordu
NR	Numer rekordu
OFMT	Format zapisu liczb na wyjściu; domyślnie `%.6g`
OFS	Separator pól na wyjściu (domyślnie spacja)
ORS	Separator rekordów na wyjściu; domyślnie `\n`
RLENGTH	Długość łańcucha dopasowanego przez funkcję `match`
RS	Separator rekordów na wejściu; domyślnie `\n`
RSTART	Pozycja początku łańcucha dopasowanego przez funkcję `match`
SUBSEP	Separator indeksów tablicy; domyślnie `\034`

Różne wersje `awk` różnią się nieco zestawem zmiennych definiowanych domyślnie i możliwych operacji.

Operatory

Operatory	Znaczenie
`= += -= *= /= %=`	Przypisanie wartości, np. `*= wyrażenie` jest równoważne `= z * (wyrażenie)`
`||`	Alternatywa (OR); *`wyrażenie1 || wyrażenie2`* jest prawdziwe, jeśli choć jedno z dwóch wyrażeń jest prawdziwe; jeśli prawdziwe jest *`wyrażenie1`*, prawdziwość *`wyrażenia2`* już nie jest sprawdzana
`&&`	Koniunkcja (AND); *`wyrażenie1 && wyrażenie2`* jest prawdziwe, jeżeli oba wyrażenia są prawdziwe
`!`	Zaprzeczenie wartości wyrażenia
`> >= < <= == != ~ !~`	Operatory relacji; operatory `~` i `!~` symbolizują dopasowanie lub jego brak
nic	Łączenie ciągów znaków
`+ -`	Dodawanie, odejmowanie
`* / % ^`	Mnożenie, dzielenie, reszta z dzielenia modulo, potęgowanie
`++ --`	Zwiększenie wartości o jeden, zmniejszenie o jeden

Instrukcje sterujące

```
if (warunek)
  instrukcja1
else
  instrukcja2
```

Jeżeli wykonywanych jest wiele operacji, można je zgrupować za pomocą nawiasów klamrowych:

```
if (warunek)
```

```
{
  instrukcja1
  instrukcja2
}
else
  instrukcja3
```

Pętla `for` ma następującą postać:

```
for (wyrażenie1; warunek; wyrażenie2)
  instrukcja
```

lub

```
for (wyrażenie1; warunek; wyrażenie2)
{
  instrukcja1
  instrukcja2
  ...
}
```

Oto postać pętli `while`:

```
while (wyrażenie)
instrukcja
```

lub

```
while (wyrażenie)
{
  instrukcja1
  instrukcja2
}
```

Tablice

W `awk` można korzystać z tablic, których elementami są liczby lub ciągi znaków. Tablice są deklarowane automatycznie w momencie użycia w programie.

Przykłady

Drukowanie wierszy w kolejności odwrotnej.

```
{ wiersz[NR] = $0 }
END {
for (i=NR; i>0; i=i-1)
  print wiersz[i]
}
```

Statystyka częstotliwości wystąpień poszczególnych słów w tekście.

```
{ for (i=1; i<=NF; i++)
  num[$i]++
}
END {
  for (slowo in num)
  print num[slowo], slowo
}
```

Utworzenie tablicy składająca się ze 100 elementów:

```
for (i=1; i<=10; i++)
for (j=1; j<=10; j++)
tablica[i,j] = 0
```

Funkcja arytmetyczne

`atan2(y,x)`	arcus tangens argumentu `y/x` w zakresie od $-\pi$ do π
`cos(x)`	cosinus
`sin(x)`	sinus
`exp(x)`	`e` (podstawa logarytmu naturalnego) do potęgi `x`
`log(x)`	logarytm naturalny
`int(x)`	część całkowita `x`; obcięcie następuje w kierunku 0
`sqrt(x)`	pierwiastek kwadratowy
`rand(x)`	liczba pseudolosowa z przedziału $0 \le x < 1$
`srand(x)`	inicjalizacja zmiennej pseudolosowej (`rand()`) przy pomocy liczby `x`

Krótkie przykłady

Drukowanie wierszy od 10 do 20, łącznie z numerem wiersza:

```
NR == 10, NR==20 { print NR, $0 }
```

Drukowanie ostatniego pola ostatniego wiersza:

```
{ pole = $NF }
END { print pole }
```

Drukowanie każdego wiersza, w którym przedostatnie pole jest mniejsze niż 5:

```
$(NF-1) < 5
```

Drukowanie łącznej liczby pól we wszystkich wierszach:

```
{ nf = nf + NF }
END { print nf }
```

Drukowanie łącznej liczby wierszy zawierającej słowo `Anna`:

```
/Anna/ { n =  n + 1 }
END { print n }
```

Drukowanie największej spośród liczb pojawiających się w pierwszym polu oraz treści zawierającego ją wiersza:

```
$1 > max { max = $1; maxline = $0 }
END { print max, maxline }
```

Drukowanie wierszy dłuższych niż 60 znaków:

```
length > 60
```

Drukowanie trzech pierwszych pól wiersza w odwróconej kolejności:

```
{ $3, $2, $1 }
```

Drukowanie wszystkich wierszy; w miejsce pierwszego pola drukowany jest numer wiersza:

```
{ $1 = NR; print }
```

Drukowanie pól każdego wiersza w odwróconej kolejności:

```
{ for (i = NF; i > 0; i = i - 1) printf("%s ", $i)
  printf("\n")
}
```

Drukowanie numerów wierszy i sumy wszystkich pól w każdym wierszu:

```
{ suma = 0
  for (i = 1; i <= NF; i = i + 1) suma = suma + $i
  print NR, suma
}
```

Do oznaczania numerów pól można użyć dowolnego wyrażenia dającego w wyniku liczbę całkowitą, na przykład:

```
BEGIN { a=1; b=2 }
{ print $(a+b) }
```

Łączenie wielu plików w jeden:

```
{ print FILENAME, $0 }
```

Rozpakowanie z powrotem do osobnych plików:

```
$1 != poprzedni { close(poprzedni); poprzedni = $i }
{ print substr($0, index($0, " ") + 1) > $1 }
```

Pierwszy wiersz w programie rozpakowującym chroni przed przekroczeniem ograniczenia na liczbę otwartych plików. Zamyka plik, który został już rozpakowany.

Rekordy wielowierszowe

Oto przykłady rekordów składających się z wielu wierszy.

Adresy

```
Jan Kowalski
ul. Sloneczna 123/5
00-999 Duze Miasto
699 999 999
J.Kowalski@mala.firma.pl
```

Bibliografia

```
Michael K. Johnson, Erik W. Troan
Linux Application Development
Addison-Wesley, Reading, Mass.
1998
```

Przyjmijmy, że mamy do czynienia z listą adresową, w której każdy rekord zawiera w pierwszych pięciu polach następujące dane: imię i nazwisko, miejsce zamieszkania (ulica i miejscowość), numer telefonu, adres e-mail. Oczywiście, informacji w każdym rekordzie może być więcej. Załóżmy, że rekordy są oddzielone pojedynczym pustym wierszem:

```
Jan Kowalski
ul. Sloneczna 123/5
00-999 Duze Miasto
699 999 999
J.Kowalski@mala.firma.pl
przedsiebiorca budowlany

Ewa Nowak
ul. Zielona 22
99-000 Miasto
66 333 4455
E.Nowak@eko-firma.pl
ksiegowa

Adam Bartkowiak
ul. Cicha 7
66-666 Male Miasto
88 121 9119
ab@superserwer.pl
student
```

Jeżeli zmiennej `RS` (jest to separator rekordów, ang. *record separator*) zostanie przypisany pusty wiersz (`RS=""`), to każda grupa wierszy staje się jednym rekordem. Wobec tego skrypt

```
BEGIN { RS = "" }
  /Miasto/
```

spowoduje wydrukowanie każdego rekordu zawierającego `Miasto`:

```
Ewa Nowak
ul. Zielona 22
99-000 Miasto
66 333 4455
E.Nowak@eko-firma.pl
ksiegowa
Adam Bartkowiak
ul. Cicha 7
66-666 Male Miasto
88 121 9119
ab@superserwer.pl
student
```

Aby rekordy drukowane na wyjściu były oddzielone od siebie pustymi wierszami, należy zdefiniować również `ORS`, separator rekordów na wyjściu,

```
BEGIN { RS = ""; ORS = "\n\n" }
/Miasto/
```

Podwójny znak nowego wiersza `\n\n` jest najprostszym separatorem pól `ORS` w tej sytuacji. Oznacza to, że rekordy wyjściowe będą oddzielone jednym pustym wierszem.
Załóżmy, że chcemy wydrukować imiona i nazwiska oraz numery telefonów wszystkich osób o nazwisku `Nowak`. Łatwo to zrobić, jeśli najpierw przypiszemy separatorowi pól znak nowego wiersza:

```
BEGIN { RS = ""; FS = "\n" }
$1 ~ /Nowak$/  { print $1, $4 }
```

Wynik tego skryptu ma następującą postać:

```
Ewa Nowak 66 333 4455
```

Gdy zmienna `RS` jest zdefiniowana jako `""`, to separator pól `FS` jest domyślnie zdefiniowany jako dowolna sekwencja spacji, tabulatorów lub znaków nowego wiersza. Gdy jednak `FS` jest jawnie zadeklarowany jako `\n`, to wtedy tylko znak nowego wiersza odgrywa rolę separatora pól.

Operacje na ciągach znaków

Następujące funkcje umożliwiają wykonywanie podstawowych operacji na ciągach znaków:

Funkcja	Znaczenie

`gsub(r,s)`	W miejsce ciągu *r* podstawia ciąg *s* w ciągu `$0`, czyli w całym bieżącym rekordzie; zwraca liczbę wykonanych podstawień
`gsub(r,s,t)`	W miejsce *r* podstawia *s* w ciągu *t*; zwraca liczbę podstawień
`index(s,t)`	Zwraca pozycję pierwszego wystąpienia ciągu *t* w ciągu *s*; zwraca 0, jeśli *t* nie pojawia się ani razu
`length(s)`	Zwraca liczbę znaków ciągu *s*
`match(s,r)`	Sprawdza, czy *s* zawiera ciąg znaków pasujący do *r*; zwraca indeks wskazujący położenie *r* w *s*; nadaje wartości zmiennym `RSTART` i `RLENGTH`
`split(s,a)`	Dzieli ciąg znaków *s* na części, które umieszcza w tablicy *a*; dzielenie następuje domyślnie w miejscach wystąpień zmiennej *FS*; zwraca liczbę powstałych pól
`split(s,a,fs)`	Dzieli ciąg znaków *s* na części, które umieszcza w tablicy *a*; dzielenie następuje w miejscach wystąpień separatora pól *fs*; zwraca liczbę powstałych pól
`sprintf(fmt,lista_wyr)`	Zwraca listę wyrażeń sformatowaną według formatu *fmt*
`sub(r,s)`	Podstawia *s* w miejsce pierwszego (od lewej strony) wystąpienia ciągu znaków zawartego w `$0`, pasującego do *r*; zwraca liczbę podstawień
`sub(r,s,t)`	Podstawia *s* w miejsce pierwszego (od lewej strony) wystąpienia ciągu znaków zawartego w *t* i pasującego do *r*; zwraca liczbę podstawień
`substr(s,p)`	Zwraca końcową część ciągu znaków *s*, poczynając od miejsca *p*
`substr(s,p,n)`	Wybiera z całego ciągu *s* podciąg o długości *n*, zaczynający się od miejsca *p*

Funkcje definiowane przez użytkownika

Użytkownik może definiować własne funkcje. Ogólna postać definicji funkcji jest następująca:

```
function nazwa(lista_parametrów) {
  instrukcje
}
```

Parametry funkcji są oddzielane przecinkami. Funkcja może zawierać instrukcję `return` powodującą wyjście z funkcji do programu wywołującego w następującej postaci:

```
return wyrażenie
```

Jeżeli instrukcja `return` nie pojawia się nigdzie w funkcji lub jeśli ostatnia wykonywana instrukcja jest inna niż `return`, to żadna wartość nie jest zwracana przez funkcję na wyjściu.

Przykład. Funkcja zwracająca większy z dwóch argumentów:

```
function max(m,n) {
  return m > n ? m : n
}
```

Instrukcje `print` i `printf`

Instrukcja `print` służy do zwykłego drukowania, a `printf` – do drukowania w postaci sformatowanej. Domyślnie instrukcja `print` drukuje zawartość zmiennej `$0` na standardowym wyjściu. Następująca instrukcja drukuje wyrażenia oddzielone od siebie znakiem (lub ciągiem znaków) przypisanym zmiennej `OFS`:

```
print wyrażenie, wyrażenie, ...
```

Wynik tej instrukcji kończy się znakiem (lub ciągiem znaków) przypisanym zmiennej `ORS`.

Wynik instrukcji drukowania może być przekazany do pliku lub do innego polecenia:

```
print wyrażenie, wyrażenie, ... > plik
print wyrażenie, wyrażenie, ... >> plik
print wyrażenie, wyrażenie, ... | polecenie
```

Instrukcje `printf` mają postać podobną do instrukcji `print`. Różnią się obecnością pierwszego argumentu, definiującego format drukowania, na przykład:

```
printf(format, wyrażenie, wyrażenie, ...)
```

Przykład. Następujące polecenie drukuje wartość dwóch zmiennych, `a` i `b`, z których pierwsza jest ciągiem znaków, a druga liczbą całkowitą:

```
printf("%s %d \n", a, b)
```

W tabeli umieszczone są symbole formatów instrukcji `printf`.

Format	Znaczenie
`C`	Znak ASCII
`D`	Liczba dziesiętna
`E`	Liczba dziesiętna w notacji wykładniczej
`F`	Liczba dziesiętna
`G`	Liczba dziesiętna zapisana w formacie `e` lub `f`; automatycznie wybrany jest format dający krótszy zapis
`O`	Liczba całkowita zapisana w notacji ósemkowej
`S`	Ciąg znaków
`X`	Liczba całkowita zapisana w notacji szesnastkowej

Instrukcja `printf("%%")` drukuje znak `%`.

Zapis do plików

Operatory przekierowania > i >> służą do kierowania wyjścia do plików, zamiast na standardowe wyjście.

Przykład.

```
$2 > 10 { print $1, $2 > "plik1" }
$2 <= 10 { print $1, $2 > "plik2" }
```

Nazwy plików muszą być podane w cudzysłowie. Nazwy plików mogą być zarówno zmiennymi jak i wyrażeniami:

```
{ print($1, $2) > ($2 > 10 ? "plik1" : "plik2") }
{ print > $1 }
```

Operator przekierowania otwiera plik tylko raz. Przy kolejnych operacjach zapisywania dane są dopisywane do otwartego pliku. Operator >> powoduje, że dotychczasowa zawartość pliku nie jest usuwana, a nowe dane są dopisywane na końcu pliku.

Wynik programu może być przekazany do innego polecenia za pomocą potoku:

```
print | polecenie
```

Przekazywanie parametrów przy wywołaniu programu

Zmienna domyślna `ARGV` jest tablicą zawierającą kolejne parametry przekazane w wierszu poleceń podczas uruchamiania programu. Zmienna `ARGC` ma wartość o jeden większą niż liczba parametrów przekazanych do programu. Na przykład, po wykonaniu polecenia

```
awk -f skrypt a 2 x
```

zmienna `ARGC` ma wartość 4, `ARGV[0]` zawiera nazwę pliku, z którego uruchomiono skrypt, `ARGV[1]` zawiera `a`, `ARGV[2]` zawiera 2, a `ARGV[3]` zawiera `x`. Podobnie, jak w przypadku języka C, nazwa polecenia wywołującego program jest przypisana zerowemu elementowi tablicy.

Przykład. Godziny odjazdu autobusów miejskich. Załóżmy, że plik `bus.dat` zawiera w kolejnych polach następujące dane: numer linii, nazwę przystanku, godziny odjazdu autobusów w formacie godzina, minuta, minuta, minuta, ... Na przykład, część dotycząca linii 87 i przystanku przy dworcu PKP mogłaby mieć następującą postać:

```
87 PKP 12 03 18 33 48
87 PKP 13 03 18 33 48
87 PKP 14 14 30 46
87 PKP 15 02 18 34 50
87 PKP 16 06 22 38 54
87 PKP 17 11 28 45
```

Następujący program drukuje godziny odjazdu wybranej linii autobusowej w określonym przedziale godzin:

```
# bus.awk
BEGIN {
   linia = ARGV[2]; godz1 = ARGV[3]; godz2 = ARGV[4]
   ARGV[2] = ""; ARGV[3] = ""; ARGV[4] = ""
   print
   print "odjazdy" " linia ", linia,":"
   print
}

{$1 == linia && $3 >= godz1 && $3 < godz2) {
   for (i=3; i<=NF; i++)
   {
      printf("%s ",$i)
   }
      print ""
   }
```

Oto przykładowe uruchomienie tego programu:

```
awk -f bus.awk bus.dat 87 13 15
```

i rezultat programu:

```
odjazdy linia 87:

 13 03 18 33 48
 14 14 30 46
```

Funkcja `system`

Funkcja `system(`*`wyrażenie`*`)` służy do wykonywania poleceń systemowych określonych wyrażeniem w nawiasie. Na przykład, następująca funkcja powoduje wyczyszczenie ekranu:

```
BEGIN { system("clear") }
```

Polecenie systemowe musi być ujęte w cudzysłów.

Skrypty powłokowe

Skrypt jest programem składającym się z poleceń interpretowanych przez powłokę, w której jest uruchomiony. Na początku pierwszego wiersza muszą być umieszczone znaki `#!`, po których należy podać pełną ścieżkę do pliku wykonywalnego powłoki lub innego narzędzia, w którym ma być interpretowany skrypt, na przykład:

```
#!/bin/sh
```

Interpreterem skryptu może być dowolna z powłok Uniksa (`/bin/sh`, `/bin/bash`, `/bin/csh`, `/bin/ksh`), intepretery innych języków skryptowych (np. `/usr/bin/perl`, `/usr/bin/python`, `/usr/bin/tcl`, `/usr/bin/awk -f`, `/usr/bin/sed -f`) lub dowolny inny plik wykonywalny (np. `/usr/bin/cat`). Materiał bieżącego rozdziału dotyczy skryptów w powłoce Bourne'a i powłok pochodnych.

Skrypt można uruchomić poleceniem `sh skrypt.sh`. Przed uruchomieniem skryptu drugim sposobem trzeba nadać uprawnienia wykonywania:

```
chmod 700 skrypt.sh
```

lub

```
chmod +x skrypt.sh
```

Teraz można uruchomić skrypt poleceniem `./skrypt.sh`.

Wszędzie poza pierwszym wierszem skryptu znak `#` oznacza początek komentarza. Wszystkie przykłady z bieżącego rozdziału dotyczą powłoki Bourne'a (`sh`, `bash`).

Instrukcja przypisania wartości zmiennej ma następującą postać:

```
$ a=wartosc
```

Wartość zmiennej można umieścić w pojedynczym lub podwójnym cudzysłowie:

```
$ a='wartosc1'
$ b="wartosc2"
```

Dzięki temu wartość zmiennej może zawierać spacje. Jeżeli prawa strona równania jest ujęta w cudzysłów ` `, jest ona najpierw interpretowana przez powłokę, po czym otrzymana wartość zostaje przypisana zmiennej, na przykład:

```
$ data=`date +%F`
$ echo $data
2003-04-18
```

Zapis `$data` oznacza wartość zmiennej `data`. Innym sposobem odwoływania się do wartości zmiennej jest notacja `${nazwazmiennej}`.

Aby się oswoić z notacją, popatrzmy na następujący przykład:

```
#!/bin/sh

echo halo
halo=Halo
echo $halo
halo="dzien dobry"
echo $halo
halo=2+2
echo $halo
```

Oto wynik działania skryptu:

```
halo
Halo
dzien dobry
2+2
```

Umieszczenie wywołania wartości zmiennej w cudzysłowie nie przeszkadza przy podstawianiu wartości. Jest to nazywane cytowaniem częściowym (ang. *partial quoting* lub *weak quoting*), w odróżnieniu od cytowania pełnego (ang. *full quoting* lub *strong quoting*), jakim jest użycie cudzysłowu pojedynczego, na przykład `'$zmienna'`. Znaczenie tej notacji ilustruje następujący skrypt:

```
#!/bin/sh
zmienna="halo"
echo $zmienna
echo "$zmienna"
echo '$zmienna'
echo \$zmienna

echo Wpisz swoje imie i nazwisko
# Instrukcja read czyta tekst ze standardowego wejścia
read zmienna

echo '$zmienna' ma teraz wartość $zmienna
```

Oto wynik działania programu:

```
halo
halo
$zmienna
$zmienna
wpisz jakis tekst
Jan Kowalski
$zmienna ma teraz wartosc Jan Kowalski
```

Niektóre zmienne są definiowane domyślnie przy każdym uruchomieniu skryptu:

Zmienna	Znaczenie

`$0`	Nazwa skryptu powłokowego
`$#`	Liczba parametrów przekazanych do skryptu
`$$`	Identyfikator procesu (PID, ang. *process identifier*) skryptu
`$?`	Stan końcowy (ang. *exit code*) ostatniego wykonanego polecenia

Poszczególne powłoki różnią się zestawem zmiennych domyślnych.

Identyfikator procesu, oznaczany symbolem `$$` można wykorzystać np. do zapisu danych w plikach tymczasowych, które następnie łatwo zidentyfikować i usunąć:

```
$ ls > tmp_$$
```

W tym przykładzie wynik polecenia `ls` został zapisany w pliku `tmp_$$`, gdzie `$$` jest pięciocyfrową liczbą oznaczającą numer procesu.

Parametry skryptu

`$1`, `$2`, ...	Wartości poszczególnych parametrów
`$*`	Lista wszystkich parametrów, traktowana jako jedna zmienna, poszczególne parametry oddzielone są znakiem separatora
`$@`	Lista wszystkich parametrów, traktowana jako jedna zmienna, poszczególne parametry nie są od siebie oddzielone

Skrypt może być uruchomiony z kilkoma parametrami, np.

```
$ ./skrypt jeden dwa trzy
```

Wówczas polecenia

```
echo $1
echo $2 $3
echo "Pelna lista parametrow: $*"
```

dadzą wynik

```
jeden
dwa trzy
jeden dwa trzy
```

Przykład. Kopiowanie *m*−*n*+1 bajtów, od bajtu *m*+1 do bajtu *n*:

```
#!/bin/bash
m=$(($3-1))
n=$4
dd if=$1 of=$2 bs=1 skip=$m count=$(($n-$m)) 2> /dev/null
```

Polecenie `dd` jest omówione w rozdziale *Przetwarzanie plików*. Pierwszy parametr skryptu jest nazwą pliku wejściowego, drugi – nazwą pliku wynikowego, trzeci – numerem bajtu, od którego należy rozpocząć kopiowanie, a czwarty – numerem ostatniego kopiowanego bajtu.

Aby skopiować 15 bajtów z *pliku1* do *pliku2*, poczynając od bajtu nr 11, należy wykonać następujące polecenie:

```
$ ./kopiuj plik1 plik2 11 25
```

Innym sposobem ustawienia parametrów jest użycie polecenia `set`:

```
set jeden dwa trzy
echo $1
echo $2 $3
```

Wynik ma następującą postać:

```
jeden
dwa trzy
```

Manipulacje związane z podstawianiem wartości zmiennych ułatwia nawias klamrowy. Dzięki temu można łączyć wartości zmiennych z łańcuchami znaków:

```
a=Adam
b=${a}" Mickiewicz"
echo $b
```

Ten skrypt daje następujący wynik:

```
Adam Mickiewicz
```

Polecenie `test`

Polecenie `test` *warunek* zwraca stan końcowy 0, jeśli sprawdzane wyrażenie jest prawdziwe, lub stan niezerowy, jeśli wyrażenie jest nieprawdziwe. przy czym prawdą w tym kontekście jest również niepusty wynik dowolnego polecenia Uniksa. Można je zapisać również w postaci `[` *warunek* `]`.

Przykład. Następujący skrypt sprawdza, czy plik istnieje i drukuje odpowiedni komunikat.

```
if test -f plik
then echo "plik istnieje"
else echo "plik nie istnieje"
fi
```

Równoważną formą instrukcji `test` jest `[ ]`, na przykład `[ -f plik ]`.

Pełna lista warunków jest przedstawiona w następującej tabeli. Wyrażenia liczbowe dotyczą liczb całkowitych.

Warunek	**Znaczenie**
`(` wyrażenie `)`	*wyrażenie* jest prawdziwe
`!` *wyrażenie*	*wyrażenie* jest fałszywe

wyrażenie1 -a *wyrażenie2*	Oba wyrażenia są prawdziwe
wyrażenie1 -o *wyrażenie2*	Prawdziwe jest *wyrażenie1* lub *wyrażenie2*
-n *łańcuch*	*łańcuch* ma długość większą od zera
-z *łańcuch*	*łańcuch* ma długość zero
łańcuch1 = *łańcuch2*	Łańcuchy są równe
łańcuch1 != *łańcuch2*	Łańcuchy się różnią
liczba1 -eq *liczba2*	Liczby są równe
liczba1 -ge *liczba2*	*liczba1* jest większa lub równa *liczba2*
liczba1 -gt *liczba2*	*liczba1* jest większa od *liczba2*
liczba1 -le *liczba2*	*liczba1* jest mniejsza lub równa *liczba2*
liczba1 -lt *liczba2*	*liczba1* jest mniejsza niż *liczba2*
liczba1 -ne *liczba2*	*liczba1* nie jest równa *liczba2*
plik1 -ef *plik1*	*plik1* i *plik2* mają ten sam numer urządzenia i ten sam numer i-węzła
plik1 -nt *plik2*	*plik1* ma późniejszą datę ostatniej modyfikacji niż *plik2*
plik1 -ot *plik2*	*plik1* jest starszy niż *plik2*
-b *plik*	*plik* istnieje i jest specjalnym plikiem blokowym
-c *plik*	*plik* istnieje i jest specjalnym plikiem znakowym
-d *plik*	*plik* istnieje i jest katalogiem
-e *plik*	*plik* istnieje
-f *plik*	*plik* istnieje i jest zwykłym plikiem
-g *plik*	*plik* istnieje i ma ustawiony bit set-group-ID
-G *plik*	*plik* istnieje, a jego grupa ma efektywny GID
-k *plik*	*plik* istnieje i ma ustawiony sticky bit
-L *plik*	*plik* istnieje i jest dowiązaniem symbolicznym
-O *plik*	*plik* istnieje, a jego właściciel ma efektywny UID
-p *plik*	*plik* istnieje i jest potokiem z przydzieloną nazwą
-r *plik*	*plik* istnieje i ma uprawnienia odczytu
-s *plik*	*plik* istnieje i ma rozmiar większy od zera
-S *plik*	*plik* istnieje i jest gniazdem (ang. *socket*)
-t [*FD*]	deskryptor pliku *FD* (domyślnie stdout) jest otwarty na terminalu
-u *plik*	*plik* istnieje i ma ustawiony bit set-user-ID
-w *plik*	*plik* istnieje i ma uprawnienia zapisu
-x *plik*	*plik* istnieje i ma uprawnienia wykonywania

Instrukcje sterujące

Szczegóły składni niektórych instrukcji w różnych powłokach mogą być różne. Na przykład w powłoce C `if` i `then` muszą znajdować się w tym samym wierszu.

if

```
if [ warunek ]
then
   polecenie1
   polecenie2
```

```
   ...
else
   polecenie3
   polecenie4
   ...
fi
```

W tym przykładzie najpierw sprawdzany jest warunek, czy plik istnieje:

```
if test -f abc
then
    ls abc
fi
```

Warunek można zapisać również z użyciem nawiasów `[]`:

```
if [ -f abc ]
then
   ...
fi

if [ -f abc ]; then
   ...
fi
```

for

```
for zmienna in lista_wartosci
do
   ...
done
```

W pętli `for` można pominąć listę wartości zmiennej. Wówczas domyślnie pętla jest wykonywana po elementach listy argumentów skryptu `$@`. Wartościami listy mogą być również nazwy plików i w takim przypadku można skorzystać ze znaków specjalnych (na przykład gwiazdki lub znaku zapytania – patrz przykład w dalszej części rozdziału). Elementy listy wartości można wypisać jawnie na początku pętli:

```
for a in kot pies 84 kon
do
   echo $a
done
```

Aby przetworzyć w pętli wszystkie pliki z bieżącego katalogu, należy użyć znaku `*`:

```
for plik in *; do
   ls -sF $plik
   sleep 3
done
```

W kolejnym przykładzie skrypt zmienia końcową część nazwy tych plików z bieżącego katalogu, które mają rozszerzenie `.xxx`. Nowe rozszerzenie ma postać `.yyy`. Pierwszy człon nazwy pozostaje bez zmian.

```
for plik in *.xxx
do
   nowy=`basename $plik .xxx`
   mv $plik $nowy.yyy
done
```

Oto sposób na połączenie w jedną całość wszystkich plików z bieżącego katalogu mających rozszerzenie `.txt`:

```
for plik in $(ls *.txt); do
  cat $plik >> calosc.txt
done
```

Pętli `for` można użyć równiez do zmiany wielkich liter w nazwach plików na małe. W tym przykładzie operacja zmiany wielkości liter wykonywana jest na wszystkich plikach:

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

while

Instrukcje pętli `while` są wykonywane, dopóki warunek jest prawdziwy.

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

W tym przykładzie wartość zmiennej `a` zwiększa się o 1 w każdej iteracji, dopóki `a` jest mniejsze lub równe 10:

```
a=1
while [ "$a" -le 10 ]
do
   echo "a="$a
   a=$(($a+1))
done
```

Czasem konieczne jest wykonywanie pętli nieskończonej `while`. W tym celu należy użyć warunku, który jest zawsze prawdziwy. Najprostszym wyrażeniem, które pełni taką rolę, jest dwukropek `:`.

```
while :
do
   instrukcje
done
```

until

Ta instrukcja nie jest dostępna we wszystkich powłokach. Nie ma jej np. w powłoce C.

```
until warunek
do
   ...
done
```

Następujący skrypt sprawdza co 20 sekund, czy zalogował się użytkownik, którego nazwa została przekazana do skryptu jako argument `$1`.

```
until who | grep "$1" > /dev/null
do
   sleep 20
done
echo "Uwaga! $1 zalogowal sie"
```

case

Instrukcja `case` umożliwia wybór jednej z wielu możliwych wartości łańcucha.

```
case zmienna in
   wzorzec1) grupa_instrukcji_1;;
   wzorzec2) grupa_instrukcji_2;;
   ...
esac
```

Oto prosty przykład dialogu z użytkownikiem:

```
x="Dzien dobry "
y="Pani"
z="Panu"
echo "Czy jesteś kobietą? [T/N]"
read $plec
case "$plec" in
  "Tak") echo "$x$y";;
  "Nie") echo "$x$z";;
  "T")   echo "$x$y";;
  "N")   echo "$x$z";;
  *)     echo "Niezrozumiala odpowiedz";;
esac
```

Wyrażenie `*` pasuje do dowolnego łańcucha i służy do obsługi odpowiedzi użytkownika, które nie są zgodne z uwzględnionymi wcześniej wariantami odpowiedzi. Zwykle umieszcza się go na końcu instrukcji `case` w celu objęcia nim wszystkich pozostałych możliwości.

Można ten przykład nieco ulepszyć, aby reakcja programu nie zależała od wielkości liter użytych przez użytkownika.

```
x="Dzien dobry "
y="Pani"
z="Panu"
echo "Czy jesteś kobietą? [T/N]"
read $plec
case "$plec" in
  "[Tt]") | "[Tt][Aa][Kk]") echo "$x$y";;
  "[Nn]") | "[Nn][Ii][Ee]") echo "$x$z";;
  *)     echo "Niezrozumiala odpowiedz";;
esac
```

Tym razem wielkość liter w odpowiedzi użytkownika nie odgrywa roli. Poprawnie zinterpretowana zostanie każda odpowiedź składająca się z jednej lub trzech liter, np. `t` lub `tAk`. Znak `|` oznacza alternatywę (lub). Zapis `[Tt]` oznacza literę `T` lub `t`.

Język C – kompilowanie programów

Kompilator języka C w systemie Unix uruchamiany jest zwykle poleceniem `cc`. Pliki wykonywalne niektórych kompilatorów mogą mieć jednak inne nazwy, na przykład `gcc` (kompilator GNU C). Bezpłatny kompilator GNU C można pobrać z katalogu `software/gcc/gcc.html w` archiwum `prep.ai.mit.edu`. Materiał tego rozdziału dotyczy kompilatora gcc. Oto poszczególne etapy działania kompilatora:

przetwarzanie wstępne (preprocessing)
kompilacja
tłumaczenie na kod maszynowy
konsolidacja

Preprocesor usuwa z plików źródłowych komentarz, wczytuje niezbędne pliki nagłówkowe `.h` zdefiniowane dyrektywą `#include`, sprawdza makra zdefiniowanie dyrektywą `#define` (lub za pomocą opcji `-D`) i wykonuje odpowiednie podstawienia.

Po przeanalizowaniu składniu kod programu jest następnie kompilowany do asemblera. Domyślnie kompilator nie tworzy pliku asemblera, który ma rozszerzenie `.s`. Można jednak takie działanie wymusić za pomocą opcji `-S`.

Następnie asembler tłumaczy kod na język maszynowy. W przypadku użycia opcji `-O` wykonywana jest także optymalizacja kodu maszynowego. W wyniku powstaje plik z rozszerzeniem `.o` i nazwie identycznej z nazwą pliku zawierającego kod źródłowy.

Program konsolidujący może być uruchamiany osobno poleceniem `ld`. W takim przypadku należy użyć opcji `-l` w celu dołączenia standardowej biblioteki języka C, `/lib/libc.a`. Jeżeli konsolidator jest uruchamiany za pomocą pliku wykonywalnego kompilatora, biblioteka `/lib/libc.a` jest dołączana automatycznie.

FAQ

Aktualna wersja dokumentu *Frequently Asked Questions* dotycząca języka C jest dostępna pod adresem `http://www.eskimo.com/~scs/C-faq/top.html` i w witrynie `http://www.faqs.org/faqs/`.

Kompilowanie

Najprostsza instrukcja kompilacji kodu w języku C ma następującą postać:

```
$ gcc -o program program.c
```

Po opcji `-o` podana jest nazwa pliku wykonywalnego. Aby utworzyć plik obiektowy z rozszerzeniem `.o`, należy wykonać następujące polecenie:

```
$ gcc -c program.c
```

Jeżeli w kodzie programu wywoływane są funkcje `f1` i `f2`, zapisane w plikach `f1.c` i `f2.c`, to polecenie kompilacji z plików źródłowych ma następującą postać:

```
$ gcc –o program program.c f1.c f2.c
```

Aby dołączyć bibliotekę, należy użyć opcji `-l`:

```
$ gcc –o program program.c f1.c f2.c –lm
```

W tym przykładzie dołączono bibliotekę standardową `libm.a`.

Jeżeli pliki `f1.c` i `f2.c` zostały już wcześniej skompilowane do plików obiektowych, to w poleceniu kompilacji należy podać ich nazwy:

```
$ gcc –o program program.c f1.o f2.o
```

Uwaga: opcje kompilatora muszą być pisane osobno, nie mogą być łączone. W następującej tabeli zawarte są niektóre z najczęściej używanych opcji kompilatora `gcc`:

Opcja	**Znaczenie**
`-ansi`	Włączenie obsługi standardu ANSI; wyłączone są rozszerzenia kompilatora `gcc` niezgodne z tym standardem
`-c` *`pliki`*	Wykonanie kompilacji bez konsolidacji i utworzenie pliku obiektowego dla każdego z plików wejściowych
`-E`	Kompilator przerywa działanie po zakończeniu pracy preprocesora; przetworzony kod jest drukowany na ekranie
`-g`	Tworzenie informacji przydatnych podczas debugowania programu
`-I`*`katalog`*	Dołączenie *`katalogu`* do listy katalogów, w których mogą znajdować się pliki dyrektywy `include`
`-L`*`katalog`*	Dołączenie *`katalogu`* do listy katalogów, w których mogą znajdować się dołączane *`biblioteki`*
`-l`*`biblioteka`*	Dołączenie *`biblioteki`*
`-o` *`plik`*	Nazwa pliku wykonywalnego; w przypadku braku tej opcji plik wykonywalny jest zapisany w pliku `a.out`
`-O` lub `-O`*`n`*	Optymalizacja kodu pod względem zajmowanej pamięci i szybkości wykonywania; opcja `-O`*`n`*, gdzie n jest równe 0, 1, 2 lub 3 (0 oznacza brak optymalizacji) i oznacza poziom optymalizacji
`-S`	Zakończenie działanie kompilatora po wykonaniu kompilacji właściwej; w wyniku powstaje plik asemblera `.s`
`-static`	Łączenie kodu programu tylko z bibliotekami statycznymi
`-v`	Wyświetlenie poleceń wykonywanych w kolejnych stadiach kompilacji
`-Wall`	Wyświetlenie dodatkowych ostrzeżeń

Tworzenie bibliotek

Oto przykład utworzenia biblioteki statycznej, składającej się z dwóch plików obiektowych:

```
$ ar rcs lib nazwa.a plik1.o plik2.o
```

Po przedrostku `lib` w nazwie pliku wynikowego należy wpisać nazwę biblioteki. Można także dodawać pojedyncze pliki obiektowe do istniejącej biblioteki:

```
$ ar rcs libnazwa.a plik3.o
```

Aby skompilować program wykorzystujący tę bibliotekę, należy użyć opcji `-L` i wpisać nazwę katalogu, w którym znajduje się plik `libnazwa.a` (w tym przykładzie jest to `/usr/local/lib`) i fragment nazwy pliku z pominięciem prefiksu `lib` i rozszerzenia `.a`:

```
$ gcc -o program program.c -L/usr/local/lib -lnazwa
```

Program `make`

Program `make`, powstał w 1975 r. w AT&T Bell Laboratories. Jego autorem jest S.I. Feldman. `make` jest generatorem poleceń. Po uruchomieniu wczytywany jest opis projektu i sprawdzane są daty ostatniej modyfikacji plików w celu określenia, które fragmenty projektu muszą być ponownie skompilowane lub przetworzone w inny sposób. Program jest uruchamiany następującym poleceniem:

```
$ make [-f makefile] [opcje] ... obiekt_docelowy ...
```

Reguły kompilacji i instrukcje są zwykle zapisane w pliku `makefile` lub `Makefile`. Wzajemne zależności między różnymi plikami większego projektu mogą być dość skomplikowane. W ramach jednego projektu można używać wielu kompilatorów, bibliotek i różnego rodzaju plików pomocniczych. Te same polecenia, zależnie od konkretnych potrzeb, mogą być wykonywane z różnymi opcjami. Stopień złożoności projektu znacznie wzrasta w przypadku współpracy wielu osób. `make` ułatwia sprawowanie kontroli nad projektem i dokonywanie zmian. Zastosowania tego narzędzia obejmują również instalowanie oprogramowania, formatowanie dokumentów i usuwania plików tymczasowych.

Oto przykład pliku `makefile`:

```
program : program.o
     cc –o program program.o
program.o : program.c
     cc –c program.c
```

W pierwszym wierszu tego przykładu zdefiniowany jest obiekt docelowy (ang. *target*). Domyślnie `make` wykonuje czynności niezbędne do utworzenia jedynie pierwszego napotkanego celu. Można jednak wybrać jeden spośród wielu celów opisanych w `makefile`. Oto przykład. `Makefile` ma następującą postać:

```
program : program.o f1.o
     cc –o program program.o f1.o
program.o : program.c
     cc –c program.c
f1.o : f1.c
     cc –c f1.c
```

Aby uaktualnić cel `f1.o`, wystarczy wykonać polecenie

```
$ make f1.o
```

Jeżeli projekt zawiera różne cele, niekoniecznie bezpośrednio od siebie zależne, warto w pliku opisowym umieścić następujący wiersz:

```
all : program1 program2
```

Aby utworzyć nowe wersje wszystkich celów określonym dyrektywą `all`, należy uruchomić `make` w następujący sposób:

```
$ make all
```

Uwaga: Jeżeli wiersz rozpoczyna się tabulatorem (`\t`), traktowany jest jako polecenie. Każde polecenie musi rozpoczynać się znakiem `\t`. Wyjątkiem od tej reguły jest rozpoczęcie polecenia w wierszu definiującym obiekt docelowy:

```
funkcja1 : funkcja1.o; cc –c funkcja1.c
```

W celu sprawdzenia, czy tabulatory znajdują się na właściwym miejscu, można użyć następującego polecenia:

```
$ cat –t –e makefile
```

Opcja `–t` powoduje wyświetlenie znaków `^I` w miejsce tabulatora, a `–e` umieszcza na końcu każdego wiersza znak `$`. Długie wiersze mogą być kontynuowane za pomocą znaku `\` na końcu wiersza. Ukośnik musi być wówczas ostatnim znakiem poprzedzającym znak `\n`:

```
program : program.o funkcja1.o funkcja2.o \
          funkcja3.o funkcja4.o
```

Znak # oznacza początek komentarza.

Ten sam obiekt docelowy może pojawiać się w wielu wierszach definiujących zależności między plikami:

```
plik.o : plik.c
     cc –c plik.c
...
plik.o : a.h b.h
```

Jeżeli pliki nie wymagają ponownego przetworzenia, wyświetlany jest komunikat, że obiekt docelowy nie wymaga aktualizacji:

```
$ make
make: 'program' is up to date.
```

Podstawowe opcje polecenia `make` przedstawione są w następującej tabeli.

Opcja	**Znaczenie**
`-b`	Nie powoduje żadnych działań; istnieje w celu utrzymania kompatybilności z innymi wersjami programu `make`
`-m`	Nie powoduje żadnych działań; istnieje w celu utrzymania kompatybilności z innymi wersjami programu `make`
`-B`	Przetwarzane są wszystkie pliki docelowe w celu uaktualnienia
`-C` *`kat`*	Przed odczytaniem pliku opisowego i wykonaniem jakichkolwiek operacji wykonane jest przejście do katalogu *`kat`*; w przypadku wielokrotnego wystąpienia opcji `–C` każda kolejna opcja `–C` jest interpretowana po uwzględnieniu poprzedniej opcji; na przykład zapis `–C / -C etc` jest

	równoważny `-C /etc`; ta opcja jest zwykle używana w rekurencyjnych wywoływaniach polecenia `make`.
`-d`	Oprócz zwykłych komunikatów drukowane mają być informacje pomocne w wykrywaniu błędów;
`-e`	Bieżące wartości zmiennych środowiskowych maja pierwszeństwo nad zmiennymi zdefiniowanymi w pliku opisowym
`-f` *`plik`*	Wskazanie nazwy pliku pełniącego rolę pliku opisowego
`-h`	Wyświetla krótkie wskazówki dotyczące uruchamiania programu
`-i`	Ignorowanie błędów w poleceniach wykonywanych zgodnie z instrukcjami pliku opisowego
`-I` *`kat`*	Nazwa katalogu, w którym znajdują się pliki dołączane dyrektywą `include`; w przypadku użycia kilku opcji `-I` katalogi są przeszukiwane w takiej kolejności, w jakiej zostały wpisane; inaczej, niż w przypadku pozostałych opcji polecenia `make`, nazwa katalogu może być wpisana bez odstępu po nazwie opcji, poprawny jest zarówno zapis `-I`*`katalog`*, jak i `-I` *`katalog`*
`-j` *`liczba`*	Określa maksymalną liczbę zadań (poleceń), jakie mogą być wykonywane równocześnie; przy wielu opcjach `-j` obowiązuje najnowsza deklaracja; pominięcie argumentu opcji oznacza brak ograniczeń na liczbę równocześnie wykonywanych zadań
`-k`	Przetwarzanie ma być kontynuowane także w przypadku wystąpienia błędu; opcja przydatna podczas wstępnego sprawdzania poprawności kompilacji i wykrywania błędów
`-l` *`poziom`*	Nowe zadania (polecenia) nie są uruchamiane, jeżeli obciążenie systemu przekracza deklarowany *`poziom`* i nie zakończyło się wykonywanie innych zadań; *`poziom`* jest liczbą zmiennoprzecinkową; pominięcie argumentu oznacza anulowanie wcześniejszych ograniczeń
`-n`	Drukowanie informacji o czynnościach, które będą wykonywane, ale bez wykonywania rzeczywistych działań
`-o` *`plik`*	Pominięcie aktualizacji pliku również wtedy, kiedy inne pliki, od których *`plik`* jest zależny, są nowsze; pominięcie także konsekwencji zmian pliku; plik jest traktowany jako bardzo stary
`-p`	Drukowanie bazy danych (reguł i wartości zmienych) stworzonej po odczytaniu *`makefile`*; aby wydrukować bazę danych bez wykonywania żadnych operacji, należy użyć opcji `-p -f/dev/null`
`-q`	Tryb „pytający”; żadne zadania nie są wykonywane; zwracany jest jedynie kod wynikowy, który jest równy zero, jeśli plik docelowy jest już uaktualniony lub ma wartość niezerową w przeciwnym razie
`-r`	Wyłączenie wbudowanych reguł domyślnych; opróżniana jest także lista przyrostków dla reguł przyrostkowych
`-R`	Wyłączenie wbudowanych zmiennych domyślnych odnoszących się do konkretnych reguł; użytkownik może nadal definiować własne zmienne domyślne; użycie tej opcji automatycznie włącza opcję `-r`
`-s`	Informacja o wykonywanych poleceniach nie jest drukowana na ekranie
`-S`	Anulowanie opcji `-k`; może mieć znaczenie jedynie w przypadku przetwarzania rekursywnego
`-t`	Uaktualnienie dat ostatniej modyfikacji plików bez wykonywania jakichkolwiek operacji – tak, jak za pomocą polecenia `touch`
`-v`	Wyświetlenie numeru wersji i informacji o prawach autorskich

`-w`	Drukowanie informacji o bieżącym katalogu przed rozpoczęciem przetwarzania i po jego zakończeniu; przydatne w bardziej złożonym przetwarzaniu rekursywnym
`-W` *`plik`*	Plik jest traktowany tak, jakby został właśnie zmodyfikowany; w połączeniu z opcją `-n` daje informację o tym, jakie zadania zostaną wykonane po wprowadzeniu zmian w *`pliku`*; jest to jedynie symulowana zmiana daty modyfikacji pliku

Jednym z działań o specjalnym znaczeniu jest `clean`. Używane jest zwykle do usunięcia zbędnych plików obiektowych `.o` powstałych w wyniku kompilacji:

```
clean :
     /bin/rm -f core *.o
```

Polecenie

```
$ make clean
```

usuwa pliki obiektowe i pliki zrzutu pamięci (ang. *core*).

Makrodefinicje

Oto przykład pliku opisowego zawierającego makrodefinicje.

```
# Wybór kompilatora
CC = gcc
# CC = cc

# Dołączane biblioteki
LDLIBS = -L/usr/local/lib/ -lmojabib

OBJ = f1.o f2.o

# Zestaw opcji przydatny podczas debugowania programów
CFLAGS = -g -Wall -ansi

# Zestaw opcji przydatny w wersji końcowej projektu
#CFLAGS = -O -Wall -ansi

# Katalog, w którym umieszczany jest plik wykonywalny
BINDIR = /usr/local/bin

program : program.o $(OBJ)
     $(CC) -o program $(OBJ) $(LDLIBS)

clean :
     rm -f $(obj) core
```

Makro jest zdefiniowane w każdym wierszu zawierającym znak =. Wartości zmiennych oznaczane są za pomocą notacji `${ }` lub `$( )`. W tym przykładzie nie wpisano reguł

tworzenia plików `program.o`, `f1.o` i `f2.o`. Jeżeli brak w pliku opisowym reguł tworzenia plików `*.o`, `make` tworzy je domyślnie z plików o tej samej nazwie z rozszerzeniem `.c`. Jest to jedna z domyślnych reguł przyrostkowych.

W ostatnim przykładzie można zdefiniować jeszcze jedno makro, modyfikując przy tym inne, które jest od niego zależne:

```
prefix = /usr/local
BINDIR = $(prefix)/bin
```

Makro `prefix` jest zwykle ścieżką do katalogu głównego projektu, którego dotyczy plik opisowy.

Niektóre makra są zdefiniowane domyślnie. Można je wydrukować poleceniem `make -p`. Są wśród nich m.in. `HOME`, `USERNAME`, `PWD`, `CC`, `SHELL`.

Plik opisowy może zawierać wywołanie innych pliku tego typu:

```
     include nazwyplików
```

Nazwy wczytywanych plików są interpretowane tak, jak w przypadku skryptów powłokowych – znaki specjalne mają to same znaczenie.

Często definiowany jest obiekt docelowy `install`, co oznacza kopiowanie plików wykonywalnych powstałych w wyniku kompilacji do odpowiednich katalogów i wykonanie innych czynności poinstalacyjnych, na przykład:

```
install :
     cp program $(BINDIR)
```

lub

```
install:
     install -m 755 program $(BINDIR)
```

Polecenie `install` kopiuje pliki do katalogu docelowego. Opcja `-m` oznacza ustawienie trybu dostępu do kopiowanych plików, analogicznie jak za pomocą polecenia `chmod`.

Indeks poleceń

`grep`	Przeszukanie pliku w celu znalezienia wierszy zgodnych z określonym wyrażeniem regularnym
`gunzip`	Przywrócenie pierwotnej postaci plików skompresowanych poleceniem `gzip`
`gzip`	Kompresja pliku za pomocą algorytmu Lempela-Ziva (LZ77)
`head`	Wyświetlenie początkowych wierszy pliku (domyślnie 10)
`hostname`	Wyświetlenie nazwy hosta
`if`	Istrukcja sterująca języku programowania powłoki
`jobs`	Wyświetlenie listy procesów uruchomionych w bieżącej powłoce
`kill`	Zamknięcie czynnego procesu
`ksh`	Powłoka Korna
`less`	Wyświetlanie pliku fragmentami mieszczącymi się w oknie terminala
`ln`	Tworzenie dowiązania symbolicznego
`lp`	Drukowanie pliku na drukarce
`lpq`	Wyświetlenie stanu kolejki drukowania
`lpr`	Drukowanie pliku na drukarce
`lprm`	Usunięcie zadania z kolejki drukowania
`lpstat`	Wyświetlenie stanu kolejki drukowania
`ls`	Drukowanie zawartości katalogu
`mail`	Prosty program obsługi poczty
`make`	Generator poleceń
`man`	Wyświetlenie opisu składni *`polecenia`*
`mkdir`	Utworzenie nowego katalogu
`more`	Wyświetlanie pliku fragmentami mieszczącymi się w oknie terminala
`mv`	Zmiana nazwy pliku (katalogu) lub przeniesienie pliku (katalogu)
`nice`	Wykonanie *`polecenia`* lub programu na niższym priorytecie
`nl`	Drukuje numery wierszy plików tekstowych lub tekstu ze standardowego wejścia
`nohup`	Wykonywanie polecenia uruchomionego za pomocą `nohup` jest kontynuowane nawet po wylogowaniu użytkownika
`od`	Drukowanie zawartości pliku na ekranie w różnych notacjach: ósemkowej, dziesiętnej, zmiennoprzecinkowej, szestnastkowej i znakowej
`passwd`	Zmiana hasła użytkownika
`paste`	Łączenie wierszy plików, wyświetlenie ich obok siebie
`pr`	Formatowanie pliku i drukowanie na standardowym wyjściu
`ps`	Wyświetlenie aktywnych procesów
`pwd`	Drukowanie pełnej ścieżki bieżącego katalogu
`rcp`	Kopiowanie z hosta zdalnego na komputer lokalny lub w odwrotnym kierunku
`renice`	Zmiana priorytetu procesu
`rlogin`	Logowanie do hosta zdalnego
`rm`	Usunięcie pliku
`rmdir`	Usunięcie katalogu
`rsh`	Uruchomienie powłoki na hoście zdalnym
`sdiff`	Wyświetlenie różnic między plikami
`sed`	Strumieniowy (nieinterakcyjny) edytor tekstu
`set`	Ustawienie lub wyświetlenie wartości zmiennych powłokowych

`sftp`	Program transferu plików oparty na protokole SSH
`sort`	Sortowanie wierszy pliku według algorytmu odpowiadającego wybranej opcji
`split`	Dzielenie pliku na zadaną liczbę fragmentów
`sh`	Powłoka Bourne'a
`ssh`	Program komunikacyjny oparty na protokole SSH
`strings`	Wyświetlenie wszystkich ciągów znaków drukowalnych, domyślnie o długości co najmniej 4 znaków, zakończonych znakiem nowego wiersza lub znakiem null. Zwykle wykorzystywane do przeglądania plików binarnych w poszukiwaniu łańcuchów ASCII
`stty`	Definiowanie parametrów terminala
`tail`	Drukowanie końcowych wierszy pliku (domyślnie 10)
`tar`	Tworzenie archiwum plików i pobieranie plików z archiwum
`tcsh`	Powłoka T-C
`tee`	Kopiowanie danych ze standardowego wyjścia do pliku
`telnet`	Program do komunikacji interakcyjnej w trybie tekstowym
`test`	Sprawdzenie prawdziwości wyrażenia
`touch`	Tworzenie pustego pliku lub uaktualnienie czasu ostatniego dostępu do istniejącego pliku
`tr`	Kopiowanie danych ze standardowego wejścia na standardowe wyjście z jednoczesną zamianą lub usunięciem niektórych znaków
`umask`	Ustawienie domyślnych uprawnień do nowo tworzonych plików
`uncompress`	Przywrócenie pierwotnej postaci plików skompresowanych poleceniem `compress`
`uniq`	Eliminowanie powtarzających się wierszy z posortowanego pliku
`until`	Instrukcja sterująca języka programowania powłoki
`uudecode`	Odtworzenie pliku zakodowanego poleceniem `uuencode`
`uuencode`	Konwersja pliku do postaci ASCII w celu przesłania go za pomocą programu `mail`
`vi`	Edytor tekstu
`w`	Wyświetlenie informacji o obciążeniu systemu, zalogowanych użytkownikach i uruchomionych przez nich programach
`wc`	Wyświetlenie informacji o liczbie wierszy, słów i znaków w pliku
`which`	Wyświetlenie nazwy pliku uruchamianego po wywołaniu *`polecenia`*
`while`	Instrukcja sterująca języka programowania powłoki
`who`	Wyświetlenie listy zalogowanych użytkowników
`whoami`	Wyświetlenie nazwy bieżącego użytkownika
`xargs`	Wykonanie polecenia (`xargs` *`polecenie ...`*) z podanymi argumentami; pozostałe argumenty są odczytywane ze standardowego wejścia
`yes`	Drukowanie w każdym wierszu na standardowym wyjściu znaku `y` (domyślnie) lub innego ciągu znaków
`zsh`	Powłoka Z, będąca rozszerzeniem powłoki Bourne'a

Tablica znaków ASCII

ASCII oznacza American Standard Code for Information Interchange. Został opublikowany w 1968 r. Jest to kod 7-bitowy. Wiele kodów 8-bitowych, m.in. ISO 8859-1, zawiera ASCII w swej dolnej połowie. Międzynarodowym standardem, będącym odpowiednikiem ASCII, jest ISO 646. Pierwsze 32 znaki i ostatni znak na tej liście są niedrukowalne.

Notacja ósemkowa	Notacja dziesiętna	Notacja szesnastkowa	Znak	Notacja specjalna w języku C
000	0	00	NUL	\0
001	1	01	SOH	
002	2	02	STX	
003	3	03	ETX	
004	4	04	EOT	
005	5	05	ENQ	
006	6	06	ACK	
007	7	07	BEL	\a
010	8	08	BS	\b
011	9	09	HT	\t
012	10	0A	LF	\n
013	11	0B	VT	\v
014	12	0C	FF	\f
015	13	0D	CR	\r
016	14	0E	SO	
017	15	0F	SI	
020	16	10	DLE	
021	17	11	DC1	
022	18	12	DC2	
023	19	13	DC3	
024	20	14	DC4	
025	21	15	NAK	
026	22	16	SYN	
027	23	17	ETB	
030	24	18	CAN	
031	25	19	EM	
032	26	1A	SUB	
033	27	1B	ESC	
034	28	1C	FS	
035	29	1D	GS	
036	30	1E	RS	
037	31	1F	US	
040	32	20	spacja	
041	33	21	!	
042	34	22	"	
043	35	23	#	
044	36	24	$	
045	37	25	%	
046	38	26	&	
047	39	27	'	
050	40	28	(	

051	41	29	)
052	42	2A	*
053	43	2B	+
054	44	2C	,
055	45	2D	-
056	46	2E	.
057	47	2F	/
060	48	30	0
061	49	31	1
062	50	32	2
063	51	33	3
064	52	34	4
065	53	35	5
066	54	36	6
067	55	37	7
070	56	38	8
071	57	39	9
072	58	3A	:
073	59	3B	;
074	60	3C	<
075	61	3D	=
076	62	3E	>
077	63	3F	?
100	64	40	@
101	65	41	A
102	66	42	B
103	67	43	C
104	68	44	D
105	69	45	E
106	70	46	F
107	71	47	G
110	72	48	H
111	73	49	I
112	74	4A	J
113	75	4B	K
114	76	4C	L
115	77	4D	M
116	78	4E	N
117	79	4F	O
120	80	50	P
121	81	51	Q
122	82	52	R
123	83	53	S
124	84	54	T
125	85	55	U
126	86	56	V
127	87	57	W
130	88	58	X
131	89	59	Y
132	90	5A	Z
133	91	5B	[

134	92	5C	\	\\
135	93	5D	]	
136	94	5E	^	
137	95	5F	_	
140	96	60	`	
141	97	61	a	
142	98	62	b	
143	99	63	c	
144	100	64	d	
145	101	65	e	
146	102	66	f	
147	103	67	g	
150	104	68	h	
151	105	69	i	
152	106	6A	j	
153	107	6B	k	
154	108	6C	l	
155	109	6D	m	
156	110	6E	n	
157	111	6F	o	
160	112	70	p	
161	113	71	q	
162	114	72	r	
163	115	73	s	
164	116	74	t	
165	117	75	u	
166	118	76	v	
167	119	77	w	
170	120	78	x	
171	121	79	y	
172	122	7A	z	
173	123	7B	{	
174	124	7C	|	
175	125	7D	}	
176	126	7E	~	
177	127	7F	DEL	

Literatura

The UNIX Programming Environment, B.W. Kernighan, R. Pike (Prentice Hall, 1984).
sed & awk, D. Dougherty, A. Robbins (O'Reilly, 1997).
Effective AWK Programming, A. Robbins (Specialized System Consultants, 1997).
The Ultimate Guide to the vi and ex Text Editors, Hewlett-Packard Company, (Benjamin/Cummings, 1990).
Combining the Bourne-shell, sed and awk in the UNIX environment for language analysis, L.M. Schmitt, K.T. Christianson, University of Aizu Technical Report, 97-2-007 (ERIC DOC No. ED 424 729 (1999)).
The New KornShell Command and Programming Language, M.I. Bolsky, D.G. Korn (Prentice Hall, 1995).
UNIX System V Release 4: An Introduction, K.H. Rosen, R.R. Rosinski, J.M. Farber, (Osborne McGraw-Hill 1990).

Witryny internetowe

Witryna	Zawartość
`www.faqs.org/faqs`	Pliki FAQ publikowane w różnych grupach dyskusyjnych
`www.freebsd.org`	Witryna systemu FreeBSD
`www.gnu.org`	Strona główna projektu GNU
`www.linuxhq.com/dist.html`	Informacje o różnych dystrybucjach Linuksa
`www.linux.org`	Witryna informacyjna na temat systemu Linux
`www.pathname.com/fhs/`	Filesystem Hierarchy Standard – struktura systemu plików w Linuksie
`www.stokely.com`	Portal z informacjami o Uniksie i odsyłaczami do zasobów internetowych; bardzo przydatny dla administratorów
`www.ugu.com`	Unix Guru Universe; zasoby dotyczące systemu Unix
`www.unix.org`	Strona główna konsorcjum The Open Group; w konsorcjum uczestniczą Fujitsu, HP, Hitachi, IBM, Sun; w tej witrynie dostępna jest m.in. bieżąca specyfikacja Single UNIX

Grupy dyskusyjne Usenet

```
comp.os.unix.*
comp.os.linux.*
pl.comp.os.unix
pl.comp.os.linux.*
```