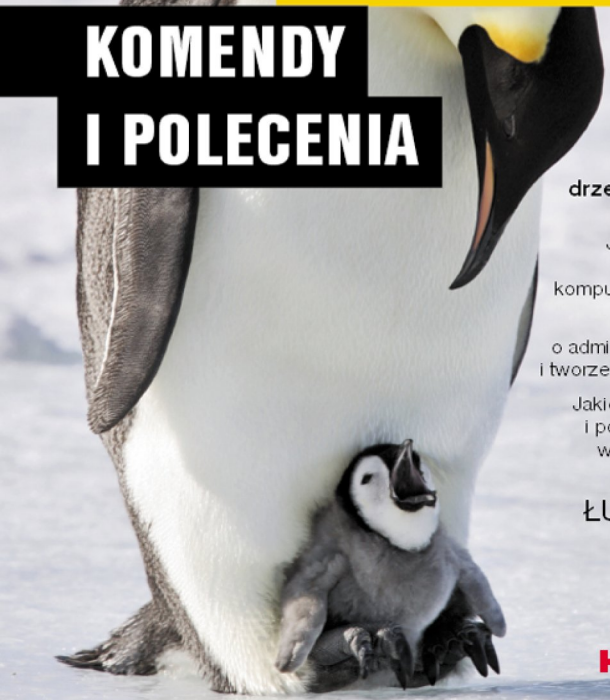
LEKSYKON KIESZONKOWY

Linux

KOMENDY I POLECENIA

Wydanie III



Odkryj wielką siłę drzemiącą w Linuksie!

Jak zainstalować Linux i zarządzać zasobami komputera z tym systemem?

Co trzeba wiedzieć o administrowaniu systemem i tworzeniu skryptów powłoki?

Jakie dodatkowe komendy i polecenia warto poznać w pierwszej kolejności?

ŁUKASZ SOSNA

Helion

Redakcja: Ewelina Burska
Projekt okładki: Maciej Pasek

Materiały graficzne na okładce zostały wykorzystane za zgodą iStockPhoto Inc.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie?linkp3_ebook
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-246-4540-4

Printed in Poland.

Spis treści

Wprowadzenie do systemu Linux 7
Czym jest Linux? 8
Dostępne dystrybucje — jak wybrać odpowiednią dla siebie? 9
Instalacja systemu 9

1. Korzystanie z komputera pracującego pod kontrolą systemu Linux 14
Środowisko pracy 14
Logowanie się do systemu 16
Bezpieczne wyłączanie i restart komputera 17
Użytkownicy systemu Linux 18
Co znajduje się w poszczególnych katalogach systemu? 19
Dyski i partycje w systemie 21
Pomoc na stronach MAN 22

2. Zarządzanie zasobami komputera 23
Pliki i katalogi w systemie 23
Wyświetlanie zawartości katalogu 24
Przechodzenie pomiędzy katalogami 34
Tworzenie katalogów 35
Usuwanie katalogów 36
Tworzenie plików 38
Usuwanie plików 38
Wyświetlenie zawartości pliku 39
Zmiana dat modyfikacji plików i dostępu do nich 40

Kopiowanie plików i katalogów 43
Przenoszenie plików i katalogów oraz zmiana ich nazwy 46
Nadawanie praw dostępu do plików i katalogów 48
Zmiana hasła 52
Zmiana powłoki 53
Uzyskiwanie informacji o typie pliku 54
Zmiana właściciela i grupy pliku 55
Wyszukiwanie plików i katalogów 56
Wypisywanie ilości bajtów, słów i linii 62
Porównywanie plików lub zakresów bajtów 63
Uzyskiwanie informacji o ilości wolnego miejsca na partycjach 64
Ustalanie, ile miejsca zajmuje plik lub katalog 65
Polecenia more i less 67
Montowanie i odmontowywanie systemów plików 68
Aktualna ścieżka, pod którą pracujemy 70
Przełączanie się na konto innego użytkownika 70
Uzyskiwanie informacji o sprzęcie 71
Przeglądanie kalendarza 75
Aktualizacja daty i czasu 77
Kontrolowanie wysyłania wiadomości 82
Wysyłanie wiadomości do innego użytkownika 82
Wysyłanie wiadomości z pliku tekstowego 83
Wysyłanie komunikatów do wszystkich sieci z pliku tekstowego 83
Pokazywanie ostatnio zalogowanych użytkowników 83
Sprawdzanie, kto jest aktualnie zalogowany na naszym komputerze 86
Informacja o tym, kto jest zalogowany do systemu 86
Sprawdzanie swojej nazwy użytkownika 86
Pokazywanie lub ustawianie nazwy hosta systemowego 87

Wyświetlanie i ustalanie parametrów interfejsu sieciowego 89
Wyszukiwanie nazwy lub adresu IP zdalnego komputera 90
Sprawdzanie, czy dana domena jest już zarejestrowana 91
Sprawdzenie dostępności hosta 91
Czas, jaki upłynął od uruchomienia systemu 92

3. Administrowanie systemem 93
Poziom uruchomienia systemu 93
Demony usług 94
Użytkownicy 96
Grupy 99
Szukanie łańcuchów w bazie whatis 99

4. Tworzenie skryptów powłoki 100
Zmienne 102
Wypisywanie tekstu na ekranie użytkownika 103
Wartości logiczne 106
Polecenie test 107
Instrukcja if 112
Instrukcja case 114
Pętla while 115
Pętla until 115
Pętla for 115
Break 116
Continue 117
Argumenty pobierane z wiersza powłoki 117

5. Polecenia dodatkowe 118
SSH 118
Historia poleceń użytych w powłoce 123
Wypisywanie pierwszych wierszy pliku 124

Wypisywanie ostatnich linii pliku 125
Uzyskiwanie informacji o trybie tworzenia nowych plików i katalogów 126
Wyświetlanie atrybutów plików i katalogów 127
Dodatkowe prawa dostępu do plików 128
Sprawdzanie dodatkowych uprawnień do plików 129
Wyszukiwanie danych w plikach 129

Skorowidz 139

Linux. Komendy i polecenia
Leksykon kieszonkowy

Wprowadzenie do systemu Linux

Witam Cię, Czytelniku, na początku trzeciego już wydania książki, w której zaprezentuję wspaniałe oprogramowanie, jakim jest system Linux, popularnie zwany też „Pingwinem" ze względu na charakterystyczne logo. Głównym tematem tej publikacji jest praca z systemem z poziomu linii poleceń; środowisko graficzne systemu Linux jest bardzo proste i przyjazne dla użytkownika, stąd też nie ma potrzeby zajmować się nim szerzej. Mam nadzieję, że po przeczytaniu tej pozycji staniesz się — podobnie jak ja — zagorzałym fanem tego systemu operacyjnego.

System Linux stał się niezwykle popularny — jak nigdy dotąd. Głównymi cechami, które czynią z niego lidera w rozwiązaniach serwerowych, jest jego cena. System operacyjny jest darmowy, można jednak zakupić standardową pomoc w jego administracji oraz najnowsze komercyjne aktualizacje.

System zawiera w prawie każdej dużej dystrybucji (mieszczącej się na płycie DVD) nie tylko najważniejsze oprogramowanie, które ze zwykłego komputera potrafi uczynić prawdziwy serwer czy to stron WWW, FTP, czy też pocztowy, a nawet wymiany plików — ale także wiele innych elementów, dzięki którym jego konfiguracja jest bardzo łatwa.

W niniejszej książce zostało omówione posługiwanie się oprogramowaniem za pomocą poleceń. Umożliwiają one zdalne administrowanie serwerem poprzez połączenie SSH. Polecenia te dadzą Ci także możliwość ustawienia wszystkich opcji. A jeżeli Twój komputer się „zawiesi", zawsze będziesz mógł uruchomić go w trybie konsoli i tam rozwiązać problem.

Czym jest Linux?

Linux, pierwotnie samodzielny system operacyjny opracowany przez Linusa Torvaldsa, obecnie stał się podstawą dla wielu innych systemów z rodziny linuksów. Wszystkie one cechują się bardzo dużymi możliwościami konfiguracji. W pewnych zastosowaniach są wręcz niezastąpione — na przykład jeżeli chcesz stworzyć serwer w sieci (w zamierzeniu Linux miał stanowić alternatywę dla drogich wersji systemu Unix, używanego właśnie w roli serwerów) lub gdy używasz starszego komputera (system ten ma małe wymagania co do sprzętu, na którym może pracować).

Jak już wspomniano, twórcą Linuksa jest Linus Torvalds, który po raz pierwszy opublikował ten system 5 października 1991 roku. Od tego czasu hobby, jakim było dlań tworzenie jądra systemu, przerodziło się w poważne wyzwanie. To właśnie opracowane przez Torvaldsa jądro jest podstawą opartych na nim systemów operacyjnych, które nazywamy dystrybucjami Linuksa. Niemal każda dystrybucja zawiera różnorakie potrzebne w pracy z komputerem programy i dodatki, dzięki którym otrzymujemy w pełni funkcjonalny system operacyjny. System jest rozwijany jako projekt z rodziny open source, co oznacza, że licencja pozwala na swobodny dostęp do kodu źródłowego, dzięki czemu można samodzielnie go poprawiać i tworzyć własne dodatkowe pakiety.

Dostępne dystrybucje — jak wybrać odpowiednią dla siebie?

Na rynku dostępnych jest wiele dystrybucji systemu Linux. To, jaką dystrybucję powinieneś wybrać, zależy głównie od tego, do czego będziesz używał swojego systemu. Bez względu na to, czy masz zamiar stworzyć serwer w internecie, serwer w wewnętrznej sieci firmy czy też używać komputera z systemem Linux w domu, musisz odpowiedzieć sobie na pytanie, jakiego dokładnie oprogramowania będziesz potrzebował.

Większość dystrybucji Linuksa powstała poza granicami naszego kraju, lecz są także i dwie polskie — PLD Linux i Aurox Linux. Oczywiście wspominam tutaj tylko o dystrybucjach pełnych, zamieszczanych na płytach CD/DVD, a nie okrojonych systemach jedno- czy kilkudyskietkowych. Dla zamieszczonego w tej książce opisu wybór konkretnej dystrybucji nie ma dużego znaczenia. Powłoka, której będziemy tutaj używać, znajduje się niemal we wszystkich dystrybucjach i jest jedną z najbardziej rozbudowanych. Jedyna różnica pomiędzy jej wersjami w różnych dystrybucjach może wynikać z braku jakichś dodatkowych programów; na pewno jednak nie dotyczy to najpopularniejszych aplikacji, a jedynie tych, które są rzadko używane. W niniejszej książce omawiam system Linux, korzystając z jednej z najpopularniejszych dystrybucji, jaką jest Mandriva Linux. Niewątpliwym jej atutem — z punktu widzenia polskiego czytelnika — jest duży stopień polonizacji środowiska pracy systemu.

Instalacja systemu

Istnieje kilka opcji instalacyjnych systemu Mandriva Linux. Można zainstalować go na przykład lokalnie na konkretnym komputerze, z uruchomionej na nim instalacyjnej płyty DVD; możemy także przeprowadzić instalację zdalną, pobierając składniki z serwera

w sieci wewnętrznej bądź za pomocą sieci internet. To ostatnie rozwiązanie nie jest jednak szczególnie zalecane, przede wszystkim ze względu na dość długi czas instalacji systemu.

W tej książce została opisana instalacja lokalna gotowej do uruchomienia płyty DVD, bądź też dyskietki startowej. System posiada instalator graficzny oraz tekstowy. Oba rodzaje programów instalacyjnych są proste w obsłudze, a wersja graficzna w coraz nowszych edycjach systemu sprowadza się w zasadzie do klikania przycisku *Dalej*. Z tej przyczyny nie będę opisywał samego instalatora i kolejnych kroków, które są w nim wykonywane; pokażę jedynie, jak uruchomić program instalacyjny, i przekażę kilka istotnych wskazówek na temat działań, jakie trzeba przedsięwziąć.

Instalacja przy użyciu płyty DVD

Instalacja ta może być przeprowadzona na większości komputerów, ponieważ ich BIOS umożliwia uruchomienie komputera ze specjalnej płyty DVD, jaką jest płyta z systemem Linux. Ta specjalna — gotowa do uruchomienia — płyta to płyta nr 1 z dystrybucją systemu.

Instalację rozpoczynamy od ustawienia w BIOS-ie opcji startu systemu z takiej płyty DVD. Następnie należy włożyć płytę do napędu i ponownie uruchomić komputer.

Instalacja przy użyciu dyskietki

Jeżeli nasz komputer nie jest w stanie uruchomić się z płyty CD/DVD (co jest mało prawdopodobne), musimy przygotować dyskietkę startową, która uruchomi program instalatora systemu. Dyskietkę taką przygotowujemy za pomocą odpowiedniego programu zamieszczonego na pierwszej płycie z dystrybucją.

Instalację systemu rozpoczynamy od włożenia czystej (czyli sformatowanej) dyskietki do stacji dyskietek oraz pierwszej płyty z systemem Linux do napędu CD/DVD-ROM. Następnie w linii poleceń systemu MS-DOS lub MS Windows wpisujemy następującą linię:

```
X:\dosutils\rawrite
```

Wykonanie tego polecenia spowoduje uruchomienie programu `rawrite` z płyty CD/DVD. Oczywiście litera `X:` oznacza tu literę przypisaną do naszego napędu CD/DVD-ROM. Program zaraz po uruchomieniu zapyta o obraz, jaki chcemy nagrać na dyskietkę. Wpisujemy: boot.img, a następnie naciskamy klawisz *Enter*. Po przygotowaniu takiej dyskietki (co potrwa chwilę) ustawiamy BIOS komputera tak, aby nasza maszyna uruchamiała się, wykorzystując w tym celu napęd dyskietek. Następnie zapisujemy zmiany wprowadzone w BIOS-ie i ponownie uruchamiamy komputer.

Wszystkie obrazy dyskietek startowych znajdują się na płycie CD/DVD z systemem w lokalizacji:

```
X:\images\
```

Oczywiście `X:` to — jak poprzednio — litera napędu CD/DVD-ROM w naszym systemie. Warto dodać, że obrazy te pozwalają na instalację systemu poprzez sieć; można w nich także znaleźć pewne dodatkowe sterowniki.

Uwagi do procesu instalacji

Po wyświetleniu ekranu z opcjami instalacji Linuksa naciskamy klawisz *Enter* i postępujemy zgodnie ze wskazówkami instalatora. Po naciśnięciu wspomnianego klawisza automatycznie uruchomiona zostanie wersja graficzna; jeżeli jednak instalator graficzny odmówi nam posłuszeństwa, powinniśmy wpisać w linii poleceń jedną z pozycji wskazanych na ekranie.

Pozycjami tymi są:

- `text` — instalacja systemu w trybie tekstowym,
- `lowres` — instalacja w trybie graficznym, ale w niskiej rozdzielczości ekranu (640×480),
- `nofb` — wyłącza tak zwany *framebuffer*, który może spowodować problemy z instalatorem graficznym,
- `expert` — instalator nie wykrywa automatycznie sprzętu,
- `linux rescue` — tryb ratunkowy instalacji.

Podczas samego procesu instalacji należy utworzyć partycje dla Linuksa, wykorzystując w tym celu wolną przestrzeń dysku; oczywiście dla Linuksa możemy także przeznaczyć cały dysk twardy. Większość użytkowników nie potrafi ocenić, ile (i jakich) partycji będą potrzebowali; w takim przypadku zalecam wybranie opcji automatycznego dopasowania partycji i ich rozmiarów przez program instalacyjny. Jeśli jednak Czytelnik chciałby ustawić partycje samodzielnie, musi utworzyć przynajmniej następujące trzy:

- / — partycja główna,
- */home* — partycja na pliki użytkowników systemu,
- */swap* — partycja wymiany.

Partycja główna, określana jako /, jest używana do przechowywania plików systemu i wszelkich programów. Z kolei partycja */home* jest przeznaczona na dane użytkowników. Możesz dowolnie określić jej wielkość w zależności od tego, ilu użytkowników będzie korzystało z systemu i ile miejsca każdy z nich będzie potrzebował.

Dodatkowa partycja */swap* to element systemu, do którego w zwykły sposób nie będziesz miał dostępu. Jest to rozszerzenie pamięci *RAM* komputera. Kiedy jakiś uruchomiony program nie jest obecnie używany, system przenosi go do tego miejsca, zwalniając część pamięci *RAM*. Partycja ta nie powinna mieć więcej niż *100 MB*.

Jeśli zdecydujemy się na taką właśnie jej wielkość, instalator prawdopodobnie poinformuje nas o błędzie, ponieważ domyślnie wielkość tej partycji ustala on, mnożąc ilość posiadanej przez nas pamięci operacyjnej przez dwa. W przypadku wyświetlenia takiego ostrzeżenia należy kliknąć przycisk *OK*; w ten sposób określamy, iż akceptujemy taką właśnie (tj. 100-megabajtową) wielkość partycji wymiany. Domyślne ustawienia instalatora wynikają z tego, że system może być instalowany na komputerach z mniejszą ilością pamięci operacyjnej; w takim przypadku istotnie plik wymiany powinien być co najmniej dwukrotnie większy niż liczba określająca ilość zainstalowanej pamięci. W nowszych komputerach nie ma jednak sensu przeznaczać na partycję wymiany więcej niż 100 MB.

W jednym z ostatnich kroków program instalacyjny zapyta o domyślny tryb uruchamiania systemu. Zalecam wybranie trybu tekstowego, jeżeli jednak Czytelnik nie czuje się na siłach, by posługiwać się komputerem bez użycia myszy, powinien wybrać środowisko graficzne.

Rozdział 1.
Korzystanie z komputera pracującego pod kontrolą systemu Linux

Komputer działający pod kontrolą Linuksa uruchamiamy dokładnie tak samo jak w przypadku innych systemów operacyjnych, czyli naciśnięciem przycisku *Power* na obudowie. Podczas ładowania systemu na ekranie monitora można śledzić kolejne etapy uruchamiania poszczególnych programów i montowania odpowiednich partycji systemu. Jeśli w momencie wyświetlania takich komunikatów, czyli podczas uruchamiania usług, naciśniemy kombinację klawiszy *Shift+I*, będziemy mogli sami zdecydować, co uruchamiać, a czego nie. Jest to bardzo przydatne, jeżeli jakiś program został źle skonfigurowany i powoduje problemy podczas startu systemu.

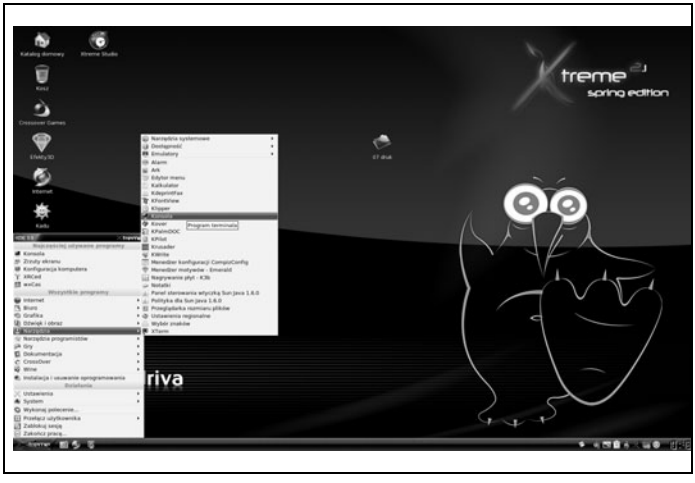
Środowisko pracy

System operacyjny taki jak Linux oferuje nam kilka środowisk pracy. Możemy podzielić je na dwie zasadnicze grupy: tekstowe i graficzne.

W środowisku graficznym wykorzystano ideę reprezentowania konkretnych poleceń systemu przez znaki w postaci ikon, poleceń *Menu* itp. Polecenie wykonywane jest po kliknięciu jego graficznej reprezentacji. To właśnie z takiego środowiska pochodzi zapewne doskonale znana większości użytkowników metoda kopiowania „przeciągnij i upuść". W Linuksie dostępnych jest kilka środowisk graficznych opartych na oprogramowaniu XFree, dla którego zostały przygotowane różne „menedżery okien", takie jak *KDE* czy *Gnome*.

Także w przypadku środowiska tekstowego w systemie dostępnych jest kilka powłok. Najpopularniejszą i najbardziej rozbudowaną jest powłoka `bash`. To właśnie ona zostanie przedstawiona w następnych rozdziałach; pokażę w nich, jak z jej pomocą zarządzać systemem i jego zasobami z poziomu linii poleceń.

Aby wykonać polecenia opisane w tej książce, trzeba skorzystać z linii poleceń. Jeżeli korzystamy z interfejsu tekstowego, nic więcej nie musimy robić. Jeżeli jednak korzystamy z interfejsu graficznego, powinniśmy uruchomić program o nazwie *Terminal*. Zrobimy to, klikając przycisk znajdujący się w lewym dolnym rogu ekranu i wybierając z rozwiniętego menu grupę *Narzędzia*, a następnie z podmenu program *Terminal*, tak jak pokazano na rysunku 1.1.



Rysunek 1.1. Uruchamianie programu Terminal

Logowanie się do systemu

Po uruchomieniu systemu musimy koniecznie zalogować się do niego. W innym wypadku korzystanie z komputera i eksplorowanie jego zasobów nie będzie możliwe.

Logowanie się w trybie tekstowym

W trybie tekstowym zobaczymy czarny ekran wraz z informacją dotyczącą tego, jakiej dystrybucji Linuksa używamy. Następnie zostaniemy poproszeni o podanie nazwy użytkownika; wpisujemy ją i zatwierdzamy naciśnięciem klawisza *Enter*. W następnym kroku pojawi się informacja o konieczności podania hasła. Podobnie jak poprzednio, wpisujemy je i zatwierdzamy klawiszem *Enter*.

```
linux login:
password:
```

Jeżeli podamy błędną nazwę użytkownika lub niepoprawne hasło, system poinformuje nas o tym i ponownie poprosi o zalogowanie się.

```
Login incorrect
linux login:
```

Jeżeli podane dane będą poprawne, zostaniemy poinformowani o tym, kiedy ostatni raz gościliśmy w systemie, po czym wyświetlony zostanie systemowy znak zachęty.

```
Last login Sat Feb 21 on tty1
[lukasz@linux /]$
```

Aby wylogować się ze swojego konta, należy użyć polecenia `logout`.

```
[lukasz@linux /]$ logout
```

Aby ponownie zalogować się na swoje konto, wpisujemy polecenie `login`, po wykonaniu którego zostaniemy poproszeni o podanie nazwy użytkownika i hasła.

```
[lukasz@linux /]$ login
```

Logowanie się w trybie graficznym

Logowanie się w trybie graficznym polega na wypełnieniu odpowiedniego formularza, w którym podajemy nazwę użytkownika oraz odpowiednie hasło. Ekran logowania w trybie graficznym wygląda tak samo we wszystkich systemach z rodziny Linuksa, poza dystrybucjami Red Hat i Aurox. W tych ostatnich systemach został zastosowany nowy interfejs graficzny, o nazwie *Bluecurve*. W zasadzie jedyna istotna różnica pomiędzy tradycyjnym i nowym ekranem logowania polega na tym, iż w pierwszym przypadku podajemy od razu nazwę użytkownika i hasło, a w interfejsie dystrybucji *Red Hat* (oraz *Aurox*) najpierw wpisujemy nazwę użytkownika, którą zatwierdzamy klawiszem *Enter*, a dopiero potem hasło, które także zatwierdzamy w ten sam sposób.

Bezpieczne wyłączanie i restart komputera

Po zalogowaniu się na swoje konto i zapoznaniu się z nowym systemem zapewne będziemy chcieli wyłączyć bezpiecznie komputer, tak aby nie uszkodzić dysku twardego. System Linux jest zaopatrzony w narzędzie, które podczas każdego uruchomienia systemu sprawdza integralność danych zapisanych na dysku twardym i to, czy nie pojawiły się na nim jakieś błędy w przypadku, gdy system został zamknięty nieprawidłowo. Aby wyłączyć komputer z poziomu linii poleceń, wpisujemy w niej polecenie `halt`.

```
[lukasz@linux /]$ halt
```

Polecenie to spowoduje zatrzymanie wszystkich działających programów i odłączanie dysków twardych, dzięki czemu nie ulegną one awarii po odcięciu napięcia zasilania. Wszystkie te operacje można będzie śledzić na ekranie. Zamiennikiem polecenia `halt` jest polecenie `shutdown` z odpowiednim parametrem:

```
[lukasz@linux /]$ shutdown -h now
```

Jeżeli natomiast chcielibyśmy ponownie uruchomić system, powinniśmy użyć polecenia `shutdown` z parametrem `-r`:

```
[lukasz@linux /]$ shutdown -r now
```

Więcej informacji na temat działania tych poleceń można znaleźć w podręczniku systemowym *MAN*.

Użytkownicy systemu Linux

Użytkownicy Linuksa mają różne prawa dostępu do zasobów komputera i prawa do uruchamiania różnych programów, tak jak w innych systemach. W każdym systemie Linux istnieje specjalne konto użytkownika o nazwie *root*; posługując się nim, można korzystać z wszystkich operacji systemowych — na konto to nie są nałożone żadne ograniczenia. Oznacza to między innymi, że taki użytkownik w wyniku swojego nieumiejętnego działania może w łatwy sposób uszkodzić system, co będzie oznaczało konieczność jego ponownej instalacji. Jeżeli zatem nie musimy pracować na tym koncie, to nigdy nie powinniśmy tego robić. Wydanie nieodpowiedniego polecenia, na przykład z błędem w zapisie, może spowodować bardzo duże straty. Powinniśmy zatem utworzyć oddzielne konto, którego będziemy używali do pracy, zaś z konta *root* będziemy korzystać tylko w przypadkach absolutnie tego wymagających.

Co znajduje się w poszczególnych katalogach systemu?

System Linux ma specyficzną strukturę dysków i założonych na nich partycji. Na pierwszy rzut oka po prostu ich nie widać. Każdy, kto pracował z systemem *Microsoft Windows* i przyzwyczaił się do okna programu *Mój komputer* i dostępnych w nim wszystkich dysków, rozpoczynając pracę w Linuksie, prawdopodobnie poczuje się nieco zagubiony. Linux wszystkie dyski, partycje i napędy „widzi" jako jedno drzewo katalogów, w którym każdy katalog rozpoczyna się od znaku /.

Wszystkie inne urządzenia, takie jak napęd dyskietek czy napędy CD/DVD-ROM, będą dostępne po ich zamontowaniu przeprowadzonym przy użyciu specjalnego polecenia. Miejscem ich montowania będzie specjalny katalog, także znajdujący się w katalogu głównym.

Na razie postaram się wymienić wszystko to, co znajduje się w katalogu głównym /. Są to następujące elementy:

- *bin* — katalog ten zawiera programy dostępne dla użytkowników systemu;
- *dev* — katalog, w którym znajdują się urządzenia blokowe i znakowe;
- *etc* — jest to katalog z plikami konfiguracyjnymi systemu;
- *home* — katalog przeznaczony na katalogi główne użytkowników systemu;
- *lib* — katalog z plikami bibliotek dla programów;
- *mnt* — katalog, w którym montowane są inne dyski;
- *proc* — katalog z informacjami o systemie;

- *root* — katalog główny użytkownika *root*;
- *sbin* — katalog z programami specjalnymi dla użytkowników o odpowiednich uprawnieniach;
- *tmp* — katalog na pliki tymczasowe;
- *usr* — katalog z programami;
- *var* — katalog używany przez system do przechowywania potrzebnych mu danych wygenerowanych przez programy lub udostępniany w części użytkownikom, na przykład na stronę WWW.

Gdzie jest miejsce na Twoje pliki?

Miejsce na dysku przeznaczone dla plików użytkownika jest dokładnie określone już podczas zakładania konta. W katalogu */home* jest tworzony osobny katalog o nazwie identycznej z nazwą konta użytkownika w systemie, w którym przechowywane są jego własne dane i pliki konfiguracyjne programów, których używa. Jeżeli np. posiadasz konto o nazwie *lukasz*, Twoim katalogiem głównym będzie */home/lukasz*. Możesz w nim wykonywać dowolne operacje, ponieważ masz do niego pełny dostęp.

Jeżeli przejrzysz zawartość katalogu głównego dla użytkowników, będziesz mógł się zorientować, ilu użytkowników korzysta z systemu. Dla każdego użytkownika została zarezerwowana pewna przestrzeń dysku twardego na jego pliki i programy. Oczywiście, Twój katalog jest dostępny tylko dla Ciebie, chyba że zmienisz uprawnienia dostępu. Podobnie jak Ty nie możesz wejść do katalogu innych użytkowników, tak i oni nie mogą zajrzeć do Twojego.

Jedynym wyjątkiem jest tutaj użytkownik *root*, czyli główny administrator systemu. Przeznaczona dla niego przestrzeń dysku znajduje się w niedostępnym dla zwykłych użytkowników katalogu */root*, ze względu na bezpieczeństwo systemu.

Dyski i partycje w systemie

Każdy system operacyjny — także Linux — ma własny sposób nazywania partycji i dysków w komputerze. W systemach Microsoft Windows wykorzystywane są do tego celu przypisane dyskom litery; w Linuksie opisy partycji i dysków odpowiadają sposobowi ich podłączenia do magistral w komputerze.

Wygląda to następująco:

- `hda` — pierwszy kontroler IDE, Master;
- `hdb` — pierwszy kontroler IDE, Slave;
- `hdc` — drugi kontroler IDE, Master;
- `hdd` — drugi kontroler IDE, Slave.

Takie oznaczenia są oczywiście stosowane tylko wtedy, gdy dyski w naszym komputerze podłączone są poprzez kanał IDE. W przypadku podłączenia poprzez interfejs SCSI nazwy dysków wyglądają podobnie:

- `sda` — pierwszy kontroler SCSI itd.

Kolejne partycje na dysku posiadają dodatkowy numer. Jeżeli utworzyliśmy partycje na dysku podpiętym do magistrali IDE, ustawionym jako urządzenie pracujące w trybie Master, to te kolejne partycje będą miały nazwy:

- `hda1`
- `hda2`
- `...`
- `hdaX`

Linux posiada także własny system plików. Obecnie stosowany jest system *ext2* i jego nowsze odmiany — *ext3* i *ext4*.

Pomoc na stronach MAN

Strony *MAN* to bogate źródło wiedzy na temat systemu Linux i posługiwania się nim za pomocą poleceń wpisywanych w terminalu. Jeżeli chcesz dowiedzieć się, do czego służy jakieś polecenie, lub uzyskać więcej szczegółowych informacji na jego temat, skorzystaj z tego podręcznika.

Aby uzyskać informacje dotyczące interesującego Cię polecenia, wpisz w linii poleceń nazwę programu `man`, a po niej samo polecenie, którego opisu potrzebujesz.

```
[lukasz@linux /]$ man find
```

Rozdział 2.
Zarządzanie zasobami komputera

W pierwszej kolejności powinieneś nauczyć się zarządzać zasobami swojego komputera — musisz nauczyć się tworzyć i usuwać elementy na dysku, zmieniać ich atrybuty oraz wykonywać wiele innych czynności, które pomogą Ci w poruszaniu się po strukturze katalogów i pozwolą w łatwy sposób dotrzeć do interesujących Cię plików.

Pliki i katalogi w systemie

W Linuksie, inaczej niż w systemach operacyjnych z rodziny Microsoft Windows, nie jest wymagane stosowanie w nazwach plików specjalnych rozszerzeń, określających m.in. to, jaki program powinien zostać użyty do otwarcia pliku. Zawartość pliku i program, jaki należy zastosować do jego otwarcia, Linux określa na podstawie nagłówka `MIME` pliku. Jeżeli jednak z jakiegokolwiek powodu odczuwasz potrzebę nadawania plikom rozszerzeń, możesz to robić — mimo że rozszerzenia te nie są konieczne, ich stosowanie nie jest także niewskazane.

W nazwach plików i katalogów możesz stosować dowolne znaki alfanumeryczne (litery i cyfry), a oprócz tego znak kropki (`.`), myślnika (`-`) i podkreślenia (`_`). Z wszystkimi innymi znakami postępuj ostrożnie; zazwyczaj są one zarezerwowane dla specjalnych funkcji systemu.

W nazwach plików i katalogów możesz używać także spacji i nie będzie to powodować większych problemów, będzie jednak po prostu niewygodne. Polecenia, w których będziesz odwoływać się do plików lub katalogów zawierających spacje, będą po prostu dłuższe i łatwiej będzie popełnić błąd podczas wpisywania nazwy pliku.

Trzeba także pamiętać o tym, że znak kropki nie powinien rozpoczynać nazwy pliku czy katalogu. Napisałem wprawdzie wcześniej, iż jego stosowanie jest dozwolone, trzeba jednak pamiętać o tym niezwykle ważnym zastrzeżeniu; nazwy plików i katalogów *nie powinny rozpoczynać się* od znaku kropki, w każdym innym jednak miejscu może on wystąpić. Przyczyną tego ograniczenia jest to, że w Linuksie znak kropki na początku nazwy pliku jest zarezerwowany dla ukrytych plików i katalogów — na przykład plik o nazwie *.ukryty_plik* będzie plikiem ukrytym.

Niezwykle istotna jest także wielkość stosowanych przez nas liter. W omawianym systemie wielkie i małe litery są rozpoznawane jako osobne znaki. Jeśli więc tworzysz katalog lub plik, zapamiętaj, czy jego nazwę wpisałeś wielką, czy małą literą — będzie to potrzebne, kiedy będziesz chciał się do niego w przyszłości odwołać.

Wyświetlany przez system znak `$` (jeśli korzystasz z konta zwykłego użytkownika)lub # (jeżeli pracujesz jako administrator systemu) jest znakiem zachęty. Oto przykład jego użycia:

```
[lukasz@linux /]$
```

Przed znakiem zachęty występują: nazwa użytkownika i nazwa hosta oraz ciąg znaków określający bieżącą lokalizację w systemie plików. W tym przypadku:

- użytkownikiem jest *lukasz*,
- host, na którym pracujemy, to *linux*,
- katalog, w którym się znajdujemy, to / (katalog główny).

Wyświetlanie zawartości katalogu

Wyświetlanie katalogów, zwane inaczej *listowaniem ich zawartości*, można wykonać w systemie Linux za pomocą kilku poleceń. System udostępnia bardzo rozbudowane polecenie służące do wykonywania tego typu zadań — jest nim `ls`.

Polecenie dir

Polecenie to może przypominać jedno z poleceń systemu operacyjnego MS-DOS — wszyscy użytkownicy, którzy znają to środowisko, zapewne poczuli się jak w domu. Jednak wynik działania tego polecenia w Linuksie różni się od tego z systemu DOS. W Linuksie wyświetlana po wykonaniu tego polecenia lista plików i katalogów jest prezentowana w postaci linii, a nie w kolumnie. Trzeba przyznać, że taki zapis utrudnia nieco odczytanie drzewa katalogów i orientację w nim, szczególnie w przypadku, gdy użytkownik przyzwyczajony jest do prezentowania go w postaci znanej z MS-DOS. Oto przykład wykonania polecenia `dir` w systemie Linux:

```
[lukasz@linux /]$ dir
bin  boot  dev  etc  home  initrd  lib  lost+found  misc
↪mnt  opt  proc  root  sbin  tmp  usr  var
```

Polecenie vdir

Wykonanie polecenia `vdir` powoduje wyświetlenie szczegółowych informacji o zawartości bieżącego katalogu; oprócz nazw plików i katalogów podawane są informacje o typie elementu, prawach dostępu do niego, jego właścicielu oraz kilka innych, które omówię na przykładzie polecenia `ls`. Oto przykład wykonania polecenia `vdir`:

```
[lukasz@linux /]$ vdir
drwxr-xr-x    2 root     root         4096 lis 28 17:47 bin
drwxr-xr-x    3 root     root         4096 lip  4  2003 boot
drwxr-xr-x   20 root     root       118784 lut 14 17:03 dev
drwxr-xr-x   62 root     root         4096 lut 14 17:03 etc
drwxr-xr-x    5 root     root         1024 wrz 13 21:07 home
drwxr-xr-x    2 root     root         4096 sty 25  2003 initrd
drwxr-xr-x    9 root     root         4096 lis 28 18:06 lib
drwx------    2 root     root        16384 lip  4  2003
↪lost+found
drwxr-xr-x    2 root     root         4096 sty 28  2003 misc
drwxr-xr-x    4 root     root         4096 lip  4  2003 mnt
```

```
drwxr-xr-x    2 root     root         4096 sty 25  2003 opt
dr-xr-xr-x   76 root     root            0 lut 14  2004 proc
drwxr-x---   22 root     root         4096 sty  5 21:01 root
drwxr-xr-x    2 root     root         8192 lip  4  2003 sbin
drwxrwxrwt   21 root     root         4096 lut 14 17:05 tmp
drwxr-xr-x   15 root     root         4096 lip  4  2003 usr
drwxr-xr-x   19 root     root         4096 lip  4  2003 var
```

Polecenie ls

Program ten jest bardzo rozbudowany i posiada wiele parametrów, które pokrótce omówię. Wykonanie samego polecenia `ls` (bez dodatkowych parametrów) da efekt identyczny z tym, jaki powodowało wykonanie omówionego poprzednio polecenia `dir`. Aby uzyskać więcej informacji na temat zawartości katalogu, powinniśmy zastosować polecenie `ls` wraz z parametrem `-l`. Wykonanie tego polecenia w takiej postaci daje wynik identyczny z wynikiem działania polecenia `vdir`.

Jako parametr można podać ścieżkę dostępu do katalogu, którego zawartość chcemy wyświetlić. Jeżeli jej nie podamy, zawsze wyświetlona zostanie zawartość katalogu bieżącego — tego, w którym się obecnie znajdujemy.

W zaprezentowanym tu przykładzie polecenie `ls` wywołane z parametrem `/etc` spowoduje wyświetlenie zawartości katalogu *etc*; użyte w drugiej linii przykładu polecenie `ls` wywołane bez parametru wyświetli natomiast zawartość katalogu bieżącego, którym jest w tym przypadku katalog główny.

```
[lukasz@linux /]$ ls /etc

[lukasz@linux /]$ ls
```

Najbardziej przydatnym parametrem polecenia `ls` jest `–l`; dzięki jego wykonaniu otrzymamy kompletny zestaw informacji na temat zawartości katalogu i typach zawartych w nim elementów. Zanim przedstawię resztę najważniejszych parametrów tego polecenia,

wyjaśnię, jak interpretować zdobyte w ten sposób informacje. Oto przykład wykonania polecenia `ls` z parametrem `-l`:

```
[lukasz@linux /]$ ls -l
drwxr-xr-x    2 root     root         4096 lis 28 17:47 bin
drwxr-xr-x    3 root     root         4096 lip  4  2003 boot
drwxr-xr-x   20 root     root       118784 lut 14 17:03 dev
drwxr-xr-x   62 root     root         4096 lut 14 17:03 etc
drwxr-xr-x    5 root     root         1024 wrz 13 21:07 home
drwxr-xr-x    2 root     root         4096 sty 25  2003 initrd
drwxr-xr-x    9 root     root         4096 lis 28 18:06 lib
drwx------    2 root     root        16384 lip  4  2003
↪lost+found
drwxr-xr-x    2 root     root         4096 sty 28  2003 misc
drwxr-xr-x    4 root     root         4096 lip  4  2003 mnt
drwxr-xr-x    2 root     root         4096 sty 25  2003 opt
dr-xr-xr-x   76 root     root            0 lut 14  2004 proc
drwxr-x---   22 root     root         4096 sty  5 21:01 root
drwxr-xr-x    2 root     root         8192 lip  4  2003 sbin
drwxrwxrwt   21 root     root         4096 lut 14 17:05 tmp
drwxr-xr-x   15 root     root         4096 lip  4  2003 usr
drwxr-xr-x   19 root     root         4096 lip  4  2003 var
```

Wyświetlone w tym przykładzie informacje, zdobyte wskutek wykonania polecenia z parametrem `-l`, zinterpretujemy na podstawie pierwszej linii wyniku:

```
drwxr-xr-x    2 root     root         4096 lis 28 17:47 bin
```

- kolumna 1.: typ elementu i prawa dostępu do niego (`drwxr-xr-x`),
- kolumna 2.: liczba powiązań do tego elementu (`2`),
- kolumna 3.: właściciel pliku (`root`),
- kolumna 4.: grupa, która została przypisana do tego pliku (`root`),
- kolumna 5.: rozmiar elementu (`4096`),
- kolumna 6.: data modyfikacji (`lis 28 17:47`),
- kolumna 7.: nazwa elementu (`bin`).

Rozpoznanie typu elementu

Zapis `drwxr-xr-x` z kolumny pierwszej składa się z czterech zasadniczych elementów. Pierwsza litera zawsze określa typ elementu.

Oto symbole oznaczające typy elementów:

- `-` — zwykły plik,
- `b` — specjalny plik blokowy,
- `c` — specjalny plik znakowy,
- `d` — katalog,
- `l` — dowiązanie symboliczne,
- `p` — nazwany potok,
- `s` — gniazdo.

A zatem, jak można wnioskować po zapisie `drwxr-xr-x`, rozpatrywany obiekt jest katalogiem.

Interpretacja praw dostępu

Niech przykładem, za pomocą którego wyjaśnię, na czym polega system praw dostępu w systemie Linux, będzie ten wiersz przykładowego wyniku wykonania polecenia `ls`, który dotyczy katalogu *var*:

```
drwxr-xr-x   19 root     root         4096 lip  4  2003 var
```

Prawa dostępu określane są tu przez litery `r`, `w` i `x`, następujące po definiującej typ elementu literze `d` (katalog). Każda litera na odpowiedniej pozycji informuje o tym, kto i jakie prawa posiada do tego pliku lub katalogu.

Zwróć uwagę na to, że w naszym przykładzie litery `x` oraz `r` występują trzykrotnie. Taki zapis określa uprawnienia według schematu: „użytkownik-grupa-inni". Litery oznaczające uprawnienia

mają różne znaczenie w zależności od tego, czy stosują się do plików, czy do katalogów.

W przypadku katalogów oznaczają następujące prawa:

- `r` — do przeszukania zawartości,
- `w` — do zmiany zawartości,
- `x` — do wejścia do katalogu.

Jakie zatem uprawnienia przypisane są do katalogu *var* z naszego przykładu? Określa je następujący zapis:

```
rwx r-x r-x
```

Oznacza to, że właściciel katalogu ma prawo do jego przeszukania, zmiany jego zawartości i wejścia do katalogu, zgodnie z zapisem `rwx`.

Grupa, która została przypisana do tego elementu, ma prawa do wejścia do katalogu i przeszukania go, zgodnie z zapisem `r-x`.

Także wszyscy inni użytkownicy mają prawo do wejścia do katalogu i przeszukania go, zgodnie z zapisem `r-x`.

Jak już wspomniałem, w przypadku plików prawa dostępu określane są przez te same symbole, jednak różna jest ich interpretacja. Tym razem litery `r`, `w` i `x` oznaczają następujące prawa:

- `r` — do odczytania pliku,
- `w` — do modyfikacji pliku,
- `x` — do uruchamiania pliku.

Rozważmy przykład z następującymi prawami dostępu do pliku:

```
rw- rw- r–
```

Powyższy zapis informuje o tym, że właściciel pliku ma prawo do jego odczytywania oraz do zmiany jego zawartości, zgodnie z zapisem `rw-`.

Także grupa, która została przypisana do pliku, ma prawo do jego odczytywania i zmiany jego zawartości, zgodnie z zapisem `rw-`.

Wszyscy inni użytkownicy mają prawo jedynie do odczytania zawartości pliku, zgodnie z zapisem `r--`.

Przejdźmy teraz do omówienia kolejnych parametrów, których można użyć z poleceniem `ls`.

`-a`

Polecenie `ls` wykonane z parametrem `-a` wyświetli wszystkie pliki i katalogi w danej lokalizacji. Pokazane zostaną także pliki ukryte, które w przypadku wywołania `ls` bez tego parametru nie są widoczne. Oto przykład wykonania `ls` z parametrem `-a`; widzimy w nim także dwa symbole: „.” (kropka) i „..” (dwie kropki), które oznaczają, odpowiednio: katalog bieżący i nadrzędny:

```
[lukasz@linux linux]$ ls -a
.  ..  katalog  plik1  plik2  plik_kopii~  .ukryty_plik
```

`-A`

Parametr ten pozwoli zobaczyć wszystkie elementy w podanej lokalizacji, wraz z plikami ukrytymi, jednak w tym przypadku w wyniku nie będą widoczne symbole „.” i „..”, które oznaczają katalog bieżący i nadrzędny:

```
[lukasz@linux linux]$ ls -A
katalog  plik1  plik2  plik_kopii~  .ukryty_plik
```

`-B`

Użycie parametru `-B` spowoduje ukrycie plików kopii zapasowych, które znajdują się w danym katalogu. Pliki te można rozpoznać po znaku tyldy (~) występującym na końcu nazwy. Jak widać, plik *plik_kopii~* nie został tutaj pokazany:

```
[lukasz@linux linux]$ ls -B
katalog  plik1  plik2
```

`-d`

Jeżeli w danym katalogu zawarte są pliki i katalogi podrzędne, polecenie `ls` wykonane bez parametru `-d` spowoduje wyświetlenie ich wszystkich. Jeżeli jednak zastosujemy parametr `-d`, zostaną wypisane tylko elementy rozpoznane jako katalogi:

```
[lukasz@linux linux]$ ls -d
katalog
```

`-I wzorzec, --ignore=wzorzec`

Dzięki temu parametrowi możemy nie pokazywać plików, których nazwy zawierają zdefiniowany przez nas wzorzec. Podobnie jak w systemie MS-DOS, także i tu możemy używać znaków ogólnych, takich jak gwiazdka (`*`), która zastępuje dowolny ciąg znaków (również pusty), oraz znak zapytania (`?`), który zastępuje dowolny pojedynczy znak.

W zaprezentowanym tu przykładzie wyświetlona została zawartość katalogu, z wyłączeniem tych elementów, których nazwy zaczynają się na literę „p”; pominięte zostały zatem *plik1*, *plik2* i *plik_kopii~*:

```
[lukasz@linux linux]$ ls --ignore='p*'
katalog
```

`wzorzec`

Dzięki podaniu wzorca nazwy elementu można wyświetlić tylko te elementy znajdujące się w danym katalogu, które pasują do tego wzorca. Wzorce tworzy się w taki sam sposób jak w systemie MS-DOS.

W tym przypadku chcemy, aby wyświetlane były elementy, których nazwa rozpoczyna się od litery „p”:

```
[lukasz@linux linux]$ ls p*
plik1  plik2  plik_kopii~
```

W kolejnym przykładzie wykorzystałem także znaki zapytania, aby lepiej zobrazować ich działanie. Taki zapis pozwala na wyświetlenie wszystkich elementów, których nazwa zaczyna się od litery „p" i składa się z pięciu znaków:

```
[lukasz@linux linux]$ ls p????
plik1  plik2
```

`-R, --recursive`

Parametry te powodują rekurencyjne wyświetlenie zawartości katalogu i jego podkatalogów. W bieżącym katalogu z naszego przykładu mamy także podkatalog o nazwie katalog. *Jak widać, po wydaniu polecenia ls z parametr*em `-R` wyświetlona została także jego zawartość:

```
[lukasz@linux linux]$ ls -R
.:
katalog  plik1  plik2  plik_kopii~

./katalog:
plik1_w_katalogu
```

`-r, --reverse`

Użycie tych parametrów powoduje odwrócenie kolejności wyświetlania zawartości katalogu. Najbardziej opcja ta przydaje się w przypadku sortowania, podczas którego możemy odwrócić domyślną kolejność wypisywanych elementów.

Dla większej jasności porównajmy zawartość przykładowego katalogu w porządku oryginalnym (bez opcji `-r`) i odwróconym (z użyciem tej opcji):

```
[lukasz@linux linux]$ ls
katalog  plik1  plik2  plik_kopii~

[lukasz@linux linux]$ ls -r
plik_kopii~  plik2  plik1  katalog
```

`-S, --sort=size`

Użycie tych parametrów powoduje posortowanie wyniku według wielkości plików. Największe pliki są wyświetlane jako pierwsze na liście, zaraz za katalogami:

```
[lukasz@linux linux]$ ls -S
katalog  duzy_plik  plik1  plik1.1  plik_kopii~
```

`-t, --sort=time`

Ten parametr sortuje wyniki według czasu ich modyfikacji. Najnowsze pliki są wyświetlane jako pierwsze:

```
[lukasz@linux linux]$ ls -t
duzy_plik  katalog  plik_kopii~  plik2  plik1
-u, --time=atime, --time=access, --time=use
```

W tym przypadku wyniki sortowane są według czasu ostatniego dostępu do pliku. Pliki, do których odwołano się ostatnio, są wyświetlane jako pierwsze.

```
[lukasz@linux linux]$ ls -u
duzy_plik  plik1  plik2  katalog  plik_kopii~
```

`-U, --sort=none`

Użycie tego parametru zapewni, że zawartość katalogu nie będzie sortowana. Elementy wyświetlone na liście wystąpią na niej dokładnie w takiej kolejności, w jakiej znajdują się w katalogu.

```
[lukasz@linux linux]$ ls -U
plik1  plik2  katalog  duzy_plik  plik_kopii~
```

`-X, --sort=extension`

Użycie tego parametru powoduje posortowanie plików według ich rozszerzeń. Pliki, które takowych nie posiadają, zawsze są wypisywane jako pierwsze.

```
[lukasz@linux linux]$ ls -x
duzy_plik  katalog  plik1  plik2  plik.aaa  plik.bbb
```

Przechodzenie pomiędzy katalogami

Do poruszania się w strukturze katalogów używamy polecenia `cd` wraz z parametrami, w zależności od tego, co mamy zamiar zrobić.

Najprostszym zastosowaniem tego polecenia jest użycie go bez podania jakiegokolwiek parametru. Jeżeli wpiszemy taką komendę, przejdziemy do naszego katalogu głównego:

```
[lukasz@linux lukasz]$ cd
```

Oczywiście zamiast `lukasz` powinieneś wpisać tu nazwę swojego konta.

Innym sposobem bezpośredniego przejścia do katalogu głównego jest podanie po poleceniu `cd` znaku tyldy (~). W systemie ten znak uznawany jest za katalog główny bieżącego użytkownika.

```
[lukasz@linux lukasz]$ cd ~
```

Jeżeli mamy zamiar przejść do wybranego przez nas katalogu, wpisujemy jego nazwę za poleceniem `cd`. Ścieżki dostępu mogą być względne, czyli podawane od miejsca, w którym się znajdujemy, lub bezwzględne, czyli podawane względem katalogu głównego, czyli /.

Dla przykładu załóżmy, że chcemy przejść z katalogu głównego do katalogu */var/www* systemu.

Pierwszym rozwiązaniem jest wykonanie tego zadania krok po kroku:

```
[lukasz@linux lukasz]$ cd /
[lukasz@linux /]$ cd var
[lukasz@linux var]$ cd www
```

Drugą możliwością jest przejście bezpośrednio do katalogu docelowego; wystarczy podać pełną ścieżkę dostępu — względną lub bezwzględną:

```
[lukasz@linux lukasz]$ cd ../../var/www

[lukasz@linux lukasz]$ cd /var/www
```

Możemy także zastosować parametr `..` (dwie kropki), który pozwala na przejście do katalogu nadrzędnego w stosunku do tego, w którym obecnie się znajdujemy. Przypuśćmy, że jesteśmy obecnie w katalogu głównym użytkownika */home/lukasz/*, a chcemy znaleźć się w katalogu nadrzędnym, czyli w */home/*. W tym celu wpisujemy polecenie `cd` z dwiema kropkami:

```
[lukasz@linux lukasz]$ cd ..
[lukasz@linux home]$
```

Tworzenie katalogów

Katalogi tworzymy za pomocą polecenia `mkdir`. Jako parametr podajemy nazwę nowego katalogu.

```
[lukasz@linux lukasz]$ mkdir katalog
```

W tym momencie utworzyliśmy katalog o nazwie *katalog*. Jeżeli jednak zamiast żądanego efektu na ekranie pojawi się komunikat:

```
mkdir: cannot create directory 'katalog': Plik istnieje
```

oznacza to, że plik lub katalog o nazwie podanej w poleceniu znajduje się już w bieżącej lokalizacji; w związku z tym musimy wymyślić inną nazwę, która jeszcze w tej lokalizacji nie występuje.

Omówmy teraz parametry polecenia `mkdir`.

`-m`

Parametr `-m` pozwala nadać odpowiednie prawa dostępu do danego katalogu w momencie jego tworzenia. Domyślnie, jeżeli nie podamy tego parametru, system sam określi prawa dostępu do katalogu.

Utwórzmy więc katalog o nazwie *kat1* najpierw bez tego parametru, a później z nim:

```
[lukasz@linux linux]$ mkdir kat1
[lukasz@linux linux]$ ls -l
drwxrwxr-x    2 lukasz   lukasz        1024 lut 21 13:45
↪kat1
```

Bez omawianego parametru do katalogu zostały przypisane domyślne prawa dostępu, w postaci `drwxrwxr-x`. Teraz spróbujmy utworzyć ten sam katalog z prawami tylko do wykonania:

```
[lukasz@linux linux]$ mkdir -m 111 kat1
[lukasz@linux linux]$ ls -l
d--x--x--x    2 lukasz   lukasz       1024 lut 21 13:44
↪kat1
```

Jak widzimy, teraz prawa są ustawione zgodnie z naszymi oczekiwaniami. Oczywiście, moglibyśmy przypisać prawa dostępu do katalogu później, po jego utworzeniu, ale skoro możemy zrobić to za pomocą jednego polecenia, wybierzmy tę właśnie, wygodniejszą opcję.

`-v, --verbose`
: Parametr ten wyświetla informację o tym, czy katalog został utworzony.

```
[lukasz@linux linux]$ mkdir -v kat1
mkdir: created directory ` kat1'
```

Usuwanie katalogów

Katalogi w systemie można usunąć w dwojaki sposób. Pierwszym jest użycie polecenia `rmdir`, przeznaczonego do usuwania katalogów. Utwórzmy więc katalog, który następnie usuniemy:

```
[lukasz@linux linux]$ mkdir kat1
[lukasz@linux linux]$ rmdir kat1
```

Polecenie to jednak potrafi usuwać tylko katalogi puste. Dla potwierdzenia utwórzmy nowy katalog, w nim zaś nowy plik, a następnie spróbujmy usunąć taką strukturę:

```
[lukasz@linux linux]$ mkdir kat2
[lukasz@linux linux]$ touch kat2/plik
[lukasz@linux linux]$ rmdir kat2
rmdir: `kat2': Katalog nie jest pusty
```

Jak widać, system odpowiedział, iż nie może usunąć katalogu, ponieważ nie jest on pusty. W takim przypadku potrzebujemy polecenia `rm`, które służy do usuwania plików; wraz z parametrem `-R` potrafi usunąć także katalog z dowolną zawartością.

Pamiętajmy także, że nie możemy znajdować się w katalogu, który mamy zamiar usunąć.

`--ignore-fail-on-non-empty`

Użycie tego parametru spowoduje, że nie zostaniemy poinformowani o podjętej próbie usunięcia katalogu niepustego. Skorzystajmy ponownie z utworzonego poprzednio katalogu *kat2* i spróbujmy go usunąć:

```
[lukasz@linux linux]$ rmdir kat1
[lukasz@linux linux]$ rmdir --ignore-fail-on-non-empty
↪kat2
[lukasz@linux linux]$
```

Katalog ten nie został usunięty, ponieważ nie jest pusty, mimo to system nie wyświetlił komunikatu o błędzie.

`--verbose`

Jeśli użyjemy tego parametru, zostaniemy poinformowani o tym, że katalog został pomyślnie usunięty — lub nie, jeżeli nie jest pusty.

```
[lukasz@linux linux]$ rmdir --verbose kat1
rmdir: removing directory, kat1

[lukasz@linux linux]$ rmdir --verbose kat2
rmdir: removing directory, kat2
rmdir: `kat2': Katalog nie jest pusty
```

Tworzenie plików

Pliki tworzymy zwykle za pomocą odpowiednich programów; dzięki temu każdy plik posiada własny format i zawiera dane zapisane w sposób specyficzny dla programu, w którym został utworzony. Możemy jednak stworzyć także pusty plik; w tym celu należy posłużyć się poleceniem `touch`.

```
[lukasz@linux lukasz]$ touch nowy_plik
```

Usuwanie plików

Jeśli chcemy usunąć plik, powinniśmy użyć do tego polecenia `rm` wraz z nazwą pliku jako parametrem:

```
[lukasz@linux lukasz]$ rm nowy_plik
```

Po wykonaniu tego polecenia można sprawdzić, czy dany plik rzeczywiście został usunięty; wyświetlmy po prostu zawartość katalogu, w którym znajdował się usuwany plik, używając do tego polecenia `ls -l` lub `vdir`.

`-r, -R, --recursive`
: Parametry te są niezwykle przydatne, gdyż pozwalają na rekurencyjne usuwanie całych struktur na dysku. Polecenie `rm` wykonane wraz z którymś z tych parametrów pozwoli usunąć niepuste katalogi, co było niemożliwe przy użyciu polecenia `rmdir`.

```
[lukasz@linux linux]$ rm -r katalog
```

`-f, --force`
: Usuwa pliki, nie pytając o potwierdzenie, i nie zgłasza błędów w przypadku, kiedy nie może usunąć danego elementu.

```
[lukasz@linux lukasz]$ rm -f plik
[lukasz@linux lukasz]$
```

`-i, --interactive`
: Powoduje wyświetlenie pytania, czy należy usunąć dany element. Odpowiedzi udzielamy, naciskając klawisz *y* (w celu potwierdzenia) lub jakikolwiek inny klawisz — w celu anulowania operacji usuwania pliku.

```
[lukasz@linux linux]$ rm -i plik1
rm: remove regular file `plik1'? y
```

`-v, --verbose`
: Wyświetla informacje o pliku, który został usunięty.

```
[lukasz@linux lukasz]$ rm -v plik1
removed `plik1'
```

Wyświetlenie zawartości pliku

Aby wyświetlić zawartość pliku lub kilku plików na ekranie, używamy polecenia `cat`. Polecenie to przyjmuje jako parametr plik lub listę plików oddzielonych znakami spacji i wyświetla je w takiej kolejności, w jakiej zostały podane.

Dla przykładu utwórzmy w swoim katalogu głównym dwa pliki: *plik1* i *plik2*. Do każdego z nich wpiszmy liczby odpowiadające jego numerowi. Najprościej zrobimy to, używając pokazanych tu dwóch poleceń, które omówię nieco później.

A oto i wspomniane polecenia; tworzymy dwa pliki i wpisujemy do nich zawartość:

```
[lukasz@linux lukasz]$ echo "1" > plik1
[lukasz@linux lukasz]$ echo "2" > plik2
```

Teraz wyświetlamy zawartość tych plików — najpierw pliku *plik1*, potem *plik2*, a na końcu obu naraz.

Wyświetlanie zawartości pliku *plik1*:

```
[lukasz@linux lukasz]$ cat plik1
1
```

Wyświetlanie zawartości pliku *plik2*:

```
[lukasz@linux lukasz]$ cat plik2
2
```

Wyświetlanie zawartości obu plików naraz:

```
[lukasz@linux lukasz]$ cat plik1 plik2
1
2
```

Jak widać, pożądany przez nas efekt został osiągnięty. Użyłem tu także nowego polecenia `echo`. Służy ono do wypisywania informacji na ekranie i jest używane podczas pisania programów w języku powłoki. W tym wypadku za pomocą tego polecenia wyświetliłem liczbę (`1` i `2`), a następnie przekierowałem ją do pliku za pomocą znaku `>`.

Zmiana dat modyfikacji plików i dostępu do nich

Zmiana daty dostępu do pliku oraz jego modyfikacji jest dość istotnym elementem działania programów w systemie Linux. Nieraz w celu przetestowania serwera DNS lub WWW będziesz musiał zmienić datę modyfikacji pliku, aby zmusić te serwery do odpowiedniej akcji lub odświeżenia swoich plików konfiguracyjnych.

Zmianę wspomnianych wartości umożliwia polecenie `touch`, omawiane już wcześniej jako narzędzie do tworzenia pustych plików. Jeżeli podamy w poleceniu nazwę pliku, który nie istnieje, rzeczywiście zostanie on utworzony:

```
[lukasz@linux lukasz]$ touch nowy_plik
```

Jak natomiast polecenie `touch` będzie działać z określonymi parametrami?

`-a`

Wydane wraz z tym parametrem polecenie `touch` zmienia tylko czas ostatniego dostępu do pliku.

```
[lukasz@linux linux]$ ls -l plik
-rw-r--r--    1 lukasz   lukasz      722 lut 21 13:36 plik

[lukasz@linux linux]$ touch -a plik

[lukasz@linux linux]$ ls -l plik
-rw-r--r--    1 lukasz   lukasz      722 lut 21 13:36 plik
```

`-c`

Jak już wcześniej wspomniałem, jeżeli plik, którego nazwę wpisujemy wraz z poleceniem `touch`, nie istnieje, to zostanie utworzony. Zapobiegniemy temu, stosując parametr `-c`. Jeżeli pliku o podanej nazwie nie ma, zaś polecenie `touch` zostało wydane wraz z parametrem `-c`, plik ten nie zostanie utworzony.

Oto przykład — wywołanie polecenia bez wspomnianego parametru utworzy plik:

```
[lukasz@linux linux]$ touch plik
[lukasz@linux linux]$ ls -l plik
-rw-rw-r--    1 lukasz   lukasz        0 lut 21 13:56 plik
```

Wywołanie z parametrem –c nie tworzy pliku w razie jego braku:

```
[lukasz@linux linux]$ touch -c plik
[lukasz@linux linux]$ ls -l plik
ls: plik: Nie ma takiego pliku ani katalogu
```

`-m`

Zmienia czas ostatniej modyfikacji pliku.

```
[lukasz@linux linux]$ ls -l plik
-rw-r--r--    1 lukasz   lukasz    722 lut 21 13:36 plik

[lukasz@linux linux]$ touch -m plik

[lukasz@linux linux]$ ls -l plik
-rw-r--r--    1 lukasz   lukasz    722 lut 21 13:54 plik
```

`-r plik_wzorca`

Parametr ten pozwala na podanie pliku, którego czas ma zostać pobrany i ustawiony jako czas modyfikacji tego pliku, którego czas chcemy zmienić.

```
[lukasz@linux linux]$ ls -l plik1 duzy_plik
-rw-r--r--    1 lukasz   lukasz      722 lut 21 13:54
↪duzy_plik
-rw-rw-r--    1 lukasz   lukasz        2 lut 15 13:05
↪plik1

[lukasz@linux linux]$ touch -r plik1 duzy_plik

[lukasz@linux linux]$ ls -l plik1 duzy_plik
-rw-r--r--    1 lukasz   lukasz      722 lut 15 13:05
↪duzy_plik
-rw-rw-r--    1 lukasz   lukasz        2 lut 15 13:05
↪plik1
[lukasz@linux linux]$
```

`-t czas`

Używając tego parametru, sami ustalamy czas modyfikacji pliku, za pomocą formatu `RR-MM-DD gg-mm`, gdzie:

- `RR` — rok,
- `MM` — miesiąc,
- `DD` — dzień,
- `gg` — godzina,
- `mm` — minuta.

Data *1 stycznia 2004 roku*, godzina *1:30* będzie w tym wypadku wyglądać następująco: `0401010130`.

```
[lukasz@linux linux]$ touch -t 0401010130 duzy_plik
[lukasz@linux linux]$ ls -l duzy_plik
-rw-r--r--    1 lukasz   lukasz        722 sty  1 01:30
↪duzy_plik
```

Kopiowanie plików i katalogów

W systemie Linux kopiowanie plików i katalogów wykonuje się za pomocą jednego polecenia — `cp` — wraz z odpowiednimi parametrami. Ogólnie składnia tego polecenia określa element, który zamierzamy skopiować, oraz miejsce, w którym ma się znaleźć kopiowany element:

```
[lukasz@linux linux]$ cp plik1 katalog1
```

W tym wypadku chcemy skopiować plik o nazwie *plik1* do katalogu *katalog1*.

Jeżeli nie podamy nazwy elementu źródłowego, tak jak w powyższym przykładzie, element zostanie skopiowany z taką samą nazwą. Jeżeli podamy nazwę, która ma być nadana kopii elementu, uzyskamy kopię z właściwą nazwą.

```
[lukasz@linux linux]$ cp plik1 katalog1/plik2
[lukasz@linux linux]$ ls katalog1
plik2
```

W tym wypadku chcemy skopiować plik o nazwie *plik1* do katalogu *katalog1*, ale kopia będzie nosić nazwę *plik2*.

`-b [metoda], --backup[=metoda]`
: Jeżeli kopiujemy element do danej lokalizacji, w której znajduje się już plik o identycznej nazwie, wówczas zostanie on nadpisany przez plik, który obecnie kopiujemy; spowoduje to utratę zawartości nadpisanego pliku. Jeśli jednak użyjemy parametru `-b` w analogicznym przypadku, plik istniejący już w danej lokalizacji zostanie zapisany jako kopia zapasowa.

  Spróbujmy zatem skopiować plik o nazwie *plik1* do katalogu *katalog1*, w którym istnieje już plik o tej nazwie. Jak widzimy, mamy teraz dwa takie pliki. Jeden to plik oryginalny, zapisany jako kopia, a drugi to plik przed chwilą przez nas skopiowany.

```
[lukasz@linux linux]$ cp -b plik1 katalog1
[lukasz@linux linux]$ ls katalog1
plik1  plik1~
```

`-d, --no-dereference`
Kopiuje dowiązania symboliczne zamiast elementów, na które te dowiązania wskazują.

`-f, --force`
Jeżeli w katalogu, do którego kopiujemy plik, istnieje już plik o identycznej nazwie, po użyciu tego parametru zostanie on nadpisany bez wyświetlenia jakiegokolwiek pytania.

```
[lukasz@linux linux]$ cp -f plik1 katalog1
```

`-i, --interactive`
Jeżeli w katalogu, do którego kopiujemy plik, występuje element o identycznej nazwie, system zapyta, czy chcemy go zastąpić plikiem, który kopiujemy. Odpowiadamy, naciskając klawisz *y* dla potwierdzenia lub jakikolwiek inny, aby zaprzeczyć.

```
[lukasz@linux linux]$ cp -i plik1 katalog1
cp: overwrite `katalog1/plik1'? y
```

`-l, --link`
Za pomocą tego parametru, zamiast kopiować element, tworzy się jego dowiązanie twarde.

`-p, --preserve`
Przy kopiowaniu plików każdy z nich traci swoje prawa dostępu, właściciela i grupę oraz czas utworzenia. Wartości te są ustawiane zgodnie z elementem, do którego kopiujemy plik. W przypadku zastosowania omawianej opcji oryginalne prawa i wszelkie inne dane dotyczące kopiowanego pliku są zachowywane.

`-r`
Parametr ten pozwala na kopiowanie rekurencyjne całych katalogów i ich zawartości wraz z zachowaniem ich struktury.

Skopiujmy więc niepusty *katalog2* do *katalog1*. Jak widać, struktura została zachowana:

```
[lukasz@linux linux]$ ls
katalog1  katalog2
[lukasz@linux linux]$ ls katalog1
[lukasz@linux linux]$ ls katalog2
katalog1_w_2  plik1  plik2  plik3
[lukasz@linux linux]$ cp -r katalog2 katalog1
[lukasz@linux linux]$ ls katalog1
katalog2
[lukasz@linux linux]$ ls katalog1/katalog2
katalog1_w_2  plik1  plik2  plik3
```

`-R, --recursive`
: Kopiuje rekurencyjnie katalogi i ich całe struktury, dodatkowo zachowując wszelkie elementy niebędące plikami.

`-S przyr_kopii, --suffix=przyr_kopii`
: Jeżeli kopiujemy element wraz z opcją `-b`, zostaną utworzone kopie zapasowe tych plików, które istnieją już w danej lokalizacji. Z kolei dzięki podanej tu opcji możemy dodatkowo ustalić własną nazwę sufiksu dla tych plików:

```
[lukasz@linux linux]$ cp -b -S _kopia * katalog1
[lukasz@linux linux]$ ls katalog1
plik1  plik1_kopia  plik2  plik2_kopia  plik3
plik3_kopia
```

`-v, --verbose`
: Informuje nas o każdym pliku, który został skopiowany.

```
[lukasz@linux linux]$ cp -v * katalog1
`plik1' -> `katalog1/plik1'
`plik2' -> `katalog1/plik2'
`plik3' -> `katalog1/plik3'
```

Przenoszenie plików i katalogów oraz zmiana ich nazwy

W systemie można przenosić pliki i katalogi oraz zmieniać ich nazwy. Jest to o tyle proste, że wymaga użycia tyko jednego polecenia — `mv`.

Aby zmienić nazwę pliku, używamy tego polecenia, podając za nim starą nazwę istniejącego już pliku i tę, na jaką chcemy ją zamienić. Załóżmy, że mamy plik *plik1* i chcemy zmienić jego nazwę na *plik2*.

```
[lukasz@linux lukasz]$ ls -l
-rw-rw-r--    1 lukasz   lukasz          4 lut 21 14:24 plik1
[lukasz@linux lukasz]$ mv plik1 plik2
[lukasz@linux lukasz]$ ls -l
-rw-rw-r--    1 lukasz   lukasz          4 lut 21 14:24 plik2
```

Przenosimy elementy niemal tak samo; jedyna różnica polega na tym, iż pierwszą nazwą jest element, który chcemy przenieść, a drugą element docelowy. Ten drugi musi istnieć fizycznie na dysku; w innym wypadku polecenie to zmienia nazwę pierwszego elementu. Spróbujmy przenieść katalog *kat1* do katalogu *kat2*.

```
[lukasz@linux lukasz]$ ls
 kat1  kat2
[lukasz@linux lukasz]$ mv kat1 kat2
[lukasz@linux lukasz]$ ls
 kat2
```

A oto parametry, których możemy użyć wraz z poleceniem `mv`:

`-b [metoda], --backup[=metoda]`
: Jeżeli przenosimy plik do katalogu, w którym istnieje już plik o tej samej nazwie, ten drugi zostanie nadpisany, wskutek czego możemy stracić zapisane w nim ważne dane. Aby tego uniknąć, powinniśmy użyć parametru `-b`. Jeżeli zdarzy się, że nazwa przenoszonego pliku będzie identyczna z nazwą jakiegoś innego pliku znajdującego się w docelowej

lokalizacji, zostanie utworzona kopia tego drugiego, istniejącego już pliku, a jego oryginał zostanie nadpisany przez przenoszony.

```
[lukasz@linux linux]$ mv -b plik1 katalog
[lukasz@linux linux]$ ls katalog
plik1  plik1~
```

`-f, --force`
Jeżeli w katalogu, do którego chcemy przenieść dany plik, istnieje już plik o takiej nazwie, to po użyciu tego parametru zostanie on nadpisany bez powiadamiania o tym użytkownika.

```
[lukasz@linux linux]$ mv -f plik1 katalog
[lukasz@linux linux]$
```

`-i, --interactive`
Jeżeli w katalogu, do którego chcemy przenieść plik, istnieje już plik o tej samej nazwie, dzięki użyciu tego parametru program zapyta, czy chcemy go nadpisać. Możemy zatwierdzić decyzję klawiszem *y* lub anulować — dowolnym innym.

```
[lukasz@linux linux]$ mv -i plik1 katalog
mv: overwrite `katalog/plik1'? y
```

`-S przyr_kopii, --suffix=przyr_kopii`
Tego parametru używamy wraz z parametrem `-b`. Umożliwia on zdefiniowanie własnego sufiksu kopii zapasowej pliku, który zostanie nadpisany.

```
[lukasz@linux linux]$ mv -b -S _kopia plik1 katalog
[lukasz@linux linux]$ ls katalog
plik1  plik1_kopia
```

`-u, --update`
Jeżeli w katalogu, do którego chcemy przenieść plik, istnieje już inny o takiej samej nazwie, to zostanie on nadpisany jedynie wtedy, gdy data modyfikacji przenoszonego pliku jest nowsza niż tego, który znajduje się w katalogu docelowym.

```
[lukasz@linux linux]$ mv plik1 katalog
[lukasz@linux linux]$ ls
katalog plik1
```

`-v, --verbose`
: W przypadku użycia tego parametru program informuje nas o każdym pliku, który został przeniesiony.

```
[lukasz@linux linux]$ mv -v plik1 katalog
`plik1' -> `katalog/plik1'
```

Nadawanie praw dostępu do plików i katalogów

Prawa dostępu nadajemy za pomocą polecenia `chmod`. Określamy w nim prawa dostępu do danego elementu na dysku. Można wybrać format zapisu tych praw spośród kilku dostępnych; ja preferuję (i polecam Czytelnikowi) format numeryczny. Polega on na sumowaniu liczb odpowiadających dostępowi dla danego elementu. Przykładem składni polecenia `chmod` jest zapis:

```
chmod 777 nazwa_pliku_lub_katalogu
```

Liczby *777* oznaczają odpowiednie litery, których znaczenie było omawiane wcześniej (pamiętaj, że 7 otrzymywane jest tu w wyniku sumowania składowych):

- 4 — `r`
- 2 — `w`
- 1 — `x`

Najlepiej wytłumaczyć to na przykładzie. Utwórzmy więc plik o nazwie *plik_chmod* i przypiszmy do niego określone prawa:

```
[lukasz@linux lukasz]$ touch plik_chmod
[lukasz@linux lukasz]$ chmod 777 plik_chmod
[lukasz@linux lukasz]$ vdir
-rwxrwxrwx    1 lukasz   lukasz          0 lut 15 19:10
↪plik_chmod
```

Jak widać, zostały mu przypisane prawa `rwxrwxrwx`, co oznacza, że wszystkim użytkownikom w systemie zezwolono na podejmowanie wszystkich możliwych akcji.

Aby lepiej objaśnić zasadę stosowania zapisu numerycznego, weźmy przykładowy plik z poprzedniego omówienia. Chcemy, aby właściciel pliku posiadał prawa do zapisu i odczytu oraz uruchomienia, a grupa, do której został przypisany ten plik, tylko prawa do jego odczytu.

A zatem:

- właściciel — 4+2+1 = 7,
- grupa — 4 = 4,
- inni — 4 = 4.

Teraz używamy polecenia `chmod`, nadając prawa do pliku:

```
[lukasz@linux lukasz]$ chmod 744 plik_chmod
[lukasz@linux lukasz]$ vdir
-rwxr--r--    1 lukasz   lukasz          0 lut 15 19:10
↪plik_chmod
```

Jak widać, prawa zostały ustawione poprawnie.

A oto parametry, które można dodać do polecenia `chmod`:

`c, --changes`
: Informuje nas o plikach, których prawa rzeczywiście udało się zmienić.

```
[root@linux linux]# chmod -c 666 plik1
uprawnienia `plik1' zmienione na 0666 (rw-rw-rw-)
```

`-f, --silent, --quiet`
: Pomija informacje o błędach, jakie wystąpiły podczas próby zmiany praw dostępu.

```
[lukasz@linux linux]$ chmod 666 plik2
chmod: failed to get attributes of `plik2':
↪Nie ma takiego pliku ani katalogu
[lukasz@linux linux]$ chmod -f 666 plik2
```

`-v, --verbose`

Wyświetla informacje dla każdego pliku, którego prawa chcemy zmienić, niezależnie od tego, czy operacja zmiany uprawnień zakończyła się powodzeniem.

```
[lukasz@linux linux]$ chmod -v 777 p*
nie udało się zmienić uprawnień `plik1' na 0777
↪(rwxrwxrwx)
chmod: changing permissions of `plik1': Operacja
↪niedozwolona
uprawnienia `plik2' zmienione na 0777 (rwxrwxrwx)
```

`-R, --recursive`

Jeżeli zmieniamy prawa dostępu do katalogu, to ulegają zmianie tylko informacje odnoszące się do niego samego, a nie do elementów w nim zawartych. Dzięki parametrowi `-R` możemy zmienić rekurencyjnie także prawa dostępu do wszystkich elementów znajdujących się w danym katalogu.

```
[lukasz@linux linux]$ ls -l
drwxrwxr-x    3 lukasz   lukasz     1024 lut 21 14:58 kat2
[lukasz@linux linux]$ chmod 777 kat2
[lukasz@linux linux]$ ls -l
drwxrwxrwx    3 lukasz   lukasz     1024 lut 21 14:58 kat2
[lukasz@linux linux]$ ls -l kat2
drwxrwxr-x    2 lukasz   lukasz     1024 lut 21 14:58 kat3
[lukasz@linux linux]$ chmod -R 777 kat2
[lukasz@linux linux]$ ls -l
drwxrwxrwx    3 lukasz   lukasz     1024 lut 21 14:58 kat2
[lukasz@linux linux]$ ls -l kat2
drwxrwxrwx    2 lukasz   lukasz     1024 lut 21 14:58 kat3
[lukasz@linux linux]$
```

Tworzenie dowiązań symbolicznych

Dowiązania symboliczne to coś, co w pewnym stopniu przypomina skróty w systemach Microsoft Windows. Takie dowiązanie pozwala np. szybko przedostać się do wskazanego przezeń katalogu lub uruchomić właściwy program.

Dowiązania symboliczne tworzy się poleceniem `ln` z odpowiednimi parametrami. Samo polecenie służy także do tworzenia dowiązań twardych. A oto parametry, które możemy wykorzystać:

`-s, --symbolic`
: Parametr ten pozwala na utworzenie dowiązania symbolicznego.

```
[lukasz@linux lukasz]$ ln -s linux/plik1 dowiazanie
[lukasz@linux lukasz]$ ls -l
lrwxrwxrwx    1 lukasz   lukasz         11 lut 21 16:00
↪dowiazanie -> linux/plik1
```

Jak widzimy, w przypadku długiej formy wyświetlania zawartości katalogu dowiązanie symboliczne jest opisane i wskazuje, do jakiego elementu prowadzi: `dowiazanie -> linux/plik1`.

`-v, --verbose`
: Dzięki użyciu tego parametru będziemy informowani o poprawnym utworzeniu dowiązania symbolicznego.

```
[lukasz@linux lukasz]$ ln -s -v linux/plik1 dowiazanie
tworzenie dowiązania symbolicznego `dowiazanie' do
↪`linux/plik1'
```

Pamiętajmy o tym, że dowiązanie symboliczne posiada swoją nazwę jako plik w katalogu, w którym zostało umieszczone; w związku z tym nie można tworzyć dowiązania o nazwie identycznej z istniejącym już w tym katalogu elementem.

Tworzenie aliasów

Alias to zdefiniowane przez użytkownika polecenia, które odwołują się do używanych w systemie poleceń wraz z odpowiednim parametrem. Mechanizm ten skraca czas poświęcony na wpisywanie parametrów; warto zatem z niego skorzystać, jeżeli często używamy jakiegoś polecenia z określonymi parametrami. Do tej pory omówiliśmy już dwa aliasy: `dir` i `vdir`.

Aby zorientować się, jakie aliasy są już dostępne w systemie, wywołujemy polecenie `alias` bez żadnych parametrów.

```
[root@linux root]# alias
alias cp='cp -i'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias mc='. /usr/share/mc/bin/mc-wrapper.sh'
alias mv='mv -i'
alias rm='rm -i'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias
↪--show-dot –show-tilde'
```

Alias tworzymy przez zdefiniowanie jego nazwy oraz polecenia, do którego ma się odnosić. Zdefiniujmy na przykład alias o nazwie *pokaz_kat*, który będzie równoznaczny z wpisaniem polecenia `ls –l`:

```
[root@linux root]# alias pokaz_kat="ls -l"
```

Teraz, kiedy mamy taki alias, możemy wypróbować jego działanie, wyświetlając za jego pomocą zawartość katalogu.

```
[root@linux root]# pokaz_kat /home/lukasz/linux
razem 18
-rw-rw-rw-    3 apache   apache       4327 lut 21 17:06
↪dowiazanie
drwxr-xr-x    2 root     root         7168 sty  1  1970
↪dyskietka
drwxrwxrwx    3 lukasz   lukasz       1024 lut 21 16:54 kat1
-rw-rw-rw-    3 apache   apache       4327 lut 21 17:06 plik1
-rwxrwxrwx    1 lukasz   lukasz          0 lut 21 16:23 plik2
```

Zmiana hasła

Hasło przypisane do swojego konta możemy zmieniać za pomocą polecenia `passwd`. Wywołujemy je bez żadnych parametrów i postępujemy zgodnie z instrukcjami programu.

```
[lukasz@linux lukasz]$ passwd
Changing password for user lukasz.
Changing password for lukasz
```

```
(current) UNIX password:
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

System prosi nas o podanie obecnego hasła, a potem o wpisanie nowego i jego potwierdzenie. Jeżeli wprowadzimy wszystkie dane poprawnie, system poinformuje nas o zmianie hasła dostępu do naszego konta:

```
passwd: all authentication tokens updated successfully.
```

W przypadku, gdybyśmy chcieli zmienić hasło innego użytkownika i bylibyśmy zalogowani na koncie administratora, możemy tego dokonać. Wystarczy po poleceniu wpisać nazwę użytkownika, którego hasło mamy zamiar zmienić. Możemy tego dokonać również za pomocą zwykłego konta użytkownika — musimy tylko znać obecne hasło użytkownika.

```
[lukasz@localhost ~]$ passwd lukasz
Changing password for user lukasz.
Changing password for lukasz
(current) UNIX password:
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

System prosi nas o podanie obecnego hasła, a potem o wpisanie nowego i jego potwierdzenie. Jeżeli wprowadzimy wszystkie dane poprawnie, system poinformuje nas o zmianie hasła dostępu do konta wybranego użytkownika.

Zmiana powłoki

Możemy zmienić używaną powłokę na dowolną dostępną w naszym systemie. Obecnie używamy powłoki `bash`. Aby zmienić ją na `sh`, po prostu wpiszmy po znaku zachęty następujące polecenie:

```
[root@linux root]# sh
sh-2.05b#
```

Jak widać, powłoka została zmieniona, ale tylko dla naszej obecnej sesji. Po wylogowaniu się i ponownym zalogowaniu będziemy posługiwać się naszą domyślną powłoką. Aby dokonać zmian na stałe, trzeba ustawić swoją konfigurację zmiennych globalnych.

Uzyskiwanie informacji o typie pliku

Do tej czynności potrzebujemy polecenia `file`, dzięki któremu dowiemy się, czym jest dany element. W przypadku plików możemy także pozyskać więcej informacji, takich jak standard kodowania znaków.

```
[lukasz@linux linux]$ file *
kat1:  directory
kat2:  directory
plik1: ASCII text
```

A oto opcje do wykorzystania:

`-b`

Parametr `-b` użyty w tym poleceniu powoduje, że nazwy plików nie są pokazywane.

```
[lukasz@linux linux]$ file -b *
directory
directory
ASCII text
```

`-i`

Dzięki temu parametrowi otrzymamy typ `MIME` danego pliku, czyli informację o tym, z jakim dokładnie typem elementu mamy do czynienia i jaki program może otworzyć ten element; możemy też korzystać z zawartych w nim informacji.

```
[lukasz@linux linux]$ file -i plik1
plik1: text/plain; charset=us-ascii
```

`-z`
: Zagląda do plików skompresowanych i analizuje pliki w nich zawarte.

```
[lukasz@linux linux]$ file -z arch.gz
arch.gz: ASCII text (gzip compressed data, was "plik1",
↪from Unix)
```

Zmiana właściciela i grupy pliku

Za pomocą polecenia `chown` możemy zmienić właściciela pliku oraz grupę, do której dany plik został przypisany. Polecenie to może definiować obie wartości jednocześnie lub zmieniać tylko jedną z nich:

- *właściciel* — jeżeli podamy tylko właściciela, zostanie on zmieniony, natomiast grupa pozostanie niezmieniona;
- *właściciel:grupa* — zmiana za jednym razem i właściciela pliku, i przypisanej do pliku grupy;
- *:grupa* — w przypadku takiego zapisu zmieniana jest tylko grupa przypisana do pliku, a właściciel pozostaje ten sam.

Ogólna składnia tego polecenia to deklaracja właściciela i grupy oraz nazwy pliku:

```
[root@linux linux]# ls -l
-rwxrwxrwx    2 lukasz   lukasz          0 lut 21 15:57 plik1
[root@linux linux]# chown nobody:nobody plik1
[root@linux linux]# ls -l
-rwxrwxrwx    2 nobody   nobody          0 lut 21 15:57 plik1
```

A oto opcje, które możemy wykorzystać:

`-c, --changes`
: Wypisuje informacje o plikach, dla których zmiany przebiegły pomyślnie.

```
[root@linux linux]# chown -c lukasz:lukasz plik1
właściciel `plik1' zmieniony na lukasz:lukasz
```

`-f, --silent, --quiet`
: Parametr ten sprawia, że pomijane będą informacje o tym, iż nie można zmienić właściciela lub grupy dla danego pliku.

```
[lukasz@linux linux]$ chown root:root plik1
chown: zmiana właściciela `plik1': Operacja niedozwolona

[lukasz@linux linux]$ chown -f root:root plik1
[lukasz@linux linux]$
```

`-v , --verbose`
: Informuje nas o zmianie właściciela i grupy pliku.

```
[root@linux lukasz]# chown -v nobody:nobody plik1
właściciel `plik1' zmieniony na nobody:nobody
```

`-R, --recursive`
: Jeżeli zmieniamy grupę lub właściciela danego katalogu, to zmiana dotyczy wyłącznie tego elementu. Wszystkie inne pliki i katalogi, które są w nim zawarte, pozostają niezmienione. Aby je zmienić, używamy opcji `-R`, dzięki czemu rekurencyjnie modyfikowana jest także informacja dotycząca wszystkich plików i katalogów w danym elemencie.

```
[root@linux linux]# chown nobody:nobody kat2
[root@linux linux]# ls -l
drwxrwxr-x    3 nobody   nobody     1024 lut 21 14:58 kat2
[root@linux linux]# ls -l kat2
drwxrwxr-x    2 lukasz   lukasz     1024 lut 21 14:58 kat3
[root@linux linux]# chown -R nobody:nobody kat2
[root@linux linux]# ls -l
drwxrwxr-x    3 nobody   nobody     1024 lut 21 14:58 kat2
[root@linux linux]# ls -l kat2
drwxrwxr-x    2 nobody   nobody     1024 lut 21 14:58 kat3
```

Wyszukiwanie plików i katalogów

Polecenie `find` pozwala odnaleźć nie tylko pliki i katalogi, ale także inne elementy na dysku. Jest ono warte naszej uwagi, ponieważ posiada mnóstwo parametrów, które pozwalają nam wykonać

najbardziej zaawansowane operacje wyszukiwania. Jako parametr podajemy ścieżkę, w której system ma szukać danego elementu, i opcje, jakie ten element powinien posiadać.

```
[lukasz@linux linux]$ find ./  -name 'p*'
```

Parametry:

`-depth`
: Parametr ten powoduje przetwarzanie zawartości katalogu przed sprawdzeniem samego katalogu.

`-maxdepth głębokość`
: Dzięki temu parametrowi możemy zdefiniować „głębokość" schodzenia w poszukiwaniu danego elementu, określając maksymalne zagnieżdżenie.

```
[lukasz@linux linux]$ find ./ -maxdepth 1 -name 'p*'
./plik1
./plik2

[lukasz@linux linux]$ find ./ -maxdepth 2 -name 'p*'
./kat1/plik1
./plik1
./plik2

[lukasz@linux linux]$ find ./ -maxdepth 3 -name 'p*'
./kat1/plik1
./kat1/lat2/plik1
./plik1
./plik2
```

`-mindepth głębokość`
: Dzięki temu parametrowi możemy zdefiniować „głębokość" poszukiwania, określając minimalną głębokość, do której należy zejść w poszukiwaniu danego elementu.

```
[lukasz@linux linux]$ find ./ -mindepth 2 -name 'p*'
./kat1/plik1
./kat1/lat2/plik1

[lukasz@linux linux]$ find ./ -mindepth 1 -name 'p*'
./kat1/plik1
```

```
./kat1/lat2/plik1
./plik1
./plik2
```

W kilku parametrach, które zostaną teraz omówione, występuje litera `n`. Oznacza ona wartość, którą można podstawić w jej miejsce. Możemy wpisać dowolną znaną wartość lub wartość ze znakiem + (plus — więcej niż, np. `+1` to więcej niż `1`) bądź – (minus — mniej niż, np. `–5` to mniej niż `5`).

`-amin n`

Polecenie z tym parametrem wyszukuje pliki, do których dostęp nastąpił `n` minut temu.

```
[lukasz@linux linux]$ find ./ -amin 1 -name 'p*'

[lukasz@linux linux]$ find ./ -amin -1 -name 'p*'

[lukasz@linux linux]$ find ./ -amin +1 -name 'p*'
./kat1/plik1
./kat1/lat2/plik1
./plik1
./plik2
```

`-atime n`

Dostęp do wyszukiwanego elementu nastąpił `n * 24` godziny temu.

`-cmin n`

Wyszukuje pliki, których status został zmieniony `n` minut temu.

```
[lukasz@linux linux]$ find ./ -cmin -4 -name 'p*'
./kat1/lat2/plik1
```

`-ctime n`

Wyszukuje pliki, których status został zmieniony `n * 24` godziny temu.

`-empty`

Wyszukuje pliki puste.

```
[lukasz@linux linux]$ find ./ -empty -name 'p*'
./plik2
```

`-group nazwag`
: Wyszukuje pliki należące do danej grupy.

```
[lukasz@linux linux]$ find ./ -group lukasz
./
./kat1
./kat1/plik1
./kat1/lat2
./kat1/lat2/plik1
./plik2
./.ukryty_plik
```

`-iname wzorzec`
: Wyszukuje pliki, których nazwy pasują do wzorca, jednak nie uwzględnia różnicy w wielkości liter w nazwach elementów.

`-linksn`
: Wyszukuje pliki mające `n` dowiązań.

```
[lukasz@linux linux]$ find ./ -links 2
./kat1/lat2

[lukasz@linux linux]$ find ./ -links 1
./kat1/plik1
./kat1/lat2/plik1
./plik2
./.ukryty_plik

[lukasz@linux linux]$ find ./ -links 3
./
./kat1
./plik1
./dowiazanie
```

`-mmin n`
: Wyszukuje pliki modyfikowane `n` minut temu.

```
[lukasz@linux linux]$ find ./ -mmin 10
./kat1/kat2
```

`-mtime n`
: Wyszukuje pliki modyfikowane `n * 24` godziny temu.

`-name wzorzec`
: Wyszukuje pliki, których nazwa pasuje do wzorca.

```
[lukasz@linux linux]$ find ./ -name 'p?i*'
./kat1/kat2/plik1
./kat1/plik1
./plik1
./plik2
```

`-nouser`
: Wyszukuje pliki, do których nie pasuje żaden użytkownik.

`-nogroup`
: Wyszukuje pliki, do których nie została przypisana grupa.

`-perm prawa`
: Wyszukuje pliki, dla których określono odpowiednie prawa dostępu.

```
[lukasz@linux linux]$ find ./ -perm 777
./kat1
./kat1/kat2
./plik2
```

`-perm -prawa`
: Wyszukuje pliki, dla których wszystkie bity praw zostały ustawione.

`-perm +prawa`
: Wyszukuje pliki, w których jakikolwiek z bitów praw jest ustawiony.

`-regex wzorzec`
: Wyszukuje pliki posiadające nazwę zgodną ze wzorcem. Wzorzec definiuje się zgodnie z prawami dotyczącymi wyrażeń regularnych.

`-size n[bckw]`
: Wyszukuje pliki o określonym rozmiarze — w tym przypadku jest to tyle, ile jest zajętych bloków, czyli 512 jednostek bajtowych na dysku.

```
[lukasz@linux linux]$ find ./ -size +1
./
./kat1
./kat1/kat2
```

`-type – typ pliku`

Wyszukuje elementy należące do określonego w parametrze typu (katalogi, pliki, dowiązania lub urządzenia).

```
[lukasz@linux linux]$ find ./ -type f
./kat1/kat2/plik1
./kat1/plik1
./plik1
./plik2
./.ukryty_plik
./dowiazanie

[lukasz@linux linux]$ find ./ -type d
./
./kat1
./kat1/kat2
```

Typy elementów to:

- `b` — plik blokowy,
- `c` — plik specjalny,
- `d` — katalog,
- `p` — nazwany potok,
- `f` — zwykły plik,
- `l` — dowiązanie symboliczne,
- `s` — gniazdo.

`-user unazwa`

Wyszukuje pliki, których właścicielem jest podany użytkownik.

```
[lukasz@linux linux]$ find ./ -user nobody

[lukasz@linux linux]$ find ./ -user lukasz
./
./kat1
./kat1/kat2
./kat1/kat2/plik1
./kat1/plik1
./plik2
./.ukryty_plik
```

Wypisywanie ilości bajtów, słów i linii

Polecenie `wc` powoduje wypisanie informacji zawartych wewnątrz pliku. Nie wypisuje ono jednak zawartości pliku, tylko informacje o tym, ile w pliku znajduje się bajtów, słów lub linii. Domyślnie bez użycia jakiegokolwiek parametru polecenie zwróci nam trzy wartości.

```
[lukasz@localhost ~]$ wc plik.txt
0 4 24 plik.txt
```

`-c, --bytes, --chars`
: Opcja powoduje wypisanie liczby bajtów zapisanych w pliku. Podczas jej użycia zostanie wyświetlona jedynie ta wartość.

```
[lukasz@localhost ~]$ wc -c plik.txt
24 plik.txt
```

`-w, --words`
: Opcja powoduje wypisanie liczby słów znajdujących się w pliku. Podczas jej zastosowania zostanie wyświetlona tylko liczba słów w danym pliku.

```
[lukasz@localhost ~]$ wc -w plik.txt
4 plik.txt
```

`-l, --lines`
: Opcja wypisuje liczbę linii w danym pliku. Podczas jej użycia zostanie wyświetlona tylko liczba linii, natomiast inne elementy zostaną pominięte.

```
[lukasz@localhost ~]$ wc -l plik.txt
0 plik.txt
```

`--version`
: Opcja pozwala wyświetlić informacje o wersji programu zainstalowanego na naszym komputerze.

```
[lukasz@localhost ~]$ wc --version
wc (coreutils) 5.2.1
Autorzy: Paul Rubin i David MacKenzie.
```

```
Copyright (C) 2004 Free Software Foundation, Inc.
Ten program jest darmowy; warunki kopiowania są opisane
↪w źródłach.
Autorzy nie dają ŻADNYCH gwarancji, w tym również
↪gwarancji PRZYDATNOŚCI
DO SPRZEDAŻY LUB DO KONKRETNYCH CELÓW.
```

Porównywanie plików lub zakresów bajtów

Narzędzie `cmp` służy do porównywania dwóch plików dowolnego typu i wyświetlenia rezultatu na standardowym wyjściu. W przypadku, gdy pliki są takie same, rezultat polecenia jest pusty. W przypadku, gdy następuje różnica, numerowanie wierszy, w których wystąpiły, zaczyna się od tego wiersza.

```
[lukasz@localhost ~]$ cmp plik.txt plik2.txt
plik.txt plik2.txt różnią się: bajt 6, linia 1
```

`-c, --print-chars`
: W celu wypisania znaków, które różnią się w pliku, należy użyć tej opcji. Dzięki temu uzyskamy pełen spis różnic pomiędzy plikami. Znaki sterujące pokazywane są w postaci ^ i litera, a znaki z ustawionym bardziej znaczącym bitem poprzedzane są przedrostkiem `M-`.

```
[lukasz@localhost ~]$ cmp -c plik.txt plik2.txt
plik.txt plik2.txt różnią się: bajt 6, linia 1 zawiera
↪61 1  62 2
```

`-i ile, --ignore-initial=ile`
: Ignorowanie różnic w ilości początkowych bajtów każdego pliku dokonuje się za pomocą parametru `-i`. Jeżeli plik jest krótszy od danej ilości bajtów, którą podaliśmy, wówczas polecenie nic nie wyświetli.

```
[lukasz@localhost ~]$ cmp  plik.txt plik2.txt -i 5
plik.txt plik2.txt różnią się: bajt 1, linia 1
```

`-l, --verbose`

Wypisywanie różnicy pomiędzy plikami w postaci numerów bajtów i wartości różniących się bajtów jest możliwe dzięki zastosowaniu specjalnego parametru.

```
[lukasz@localhost ~]$ cmp -l plik.txt plik2.txt
6  61  62
```

`-s, --quiet, --silent`

W celu niewypisywania wszystkich poszczególnych różnic w plikach, a jedynie zwrócenia informacji, czy pliki się różnią, należy użyć opcji `-s`.

```
[lukasz@localhost ~]$ cmp --silent plik.txt plik2.txt
```

`-v, --version`

Do wyświetlenia wersji programu oraz informacji o jego autorach oraz licencji obowiązującej na jego używanie służy parametr `-v`.

```
[lukasz@localhost ~]$ cmp -v
cmp (GNU diffutils) 2.8.7
Written by Torbjorn Granlund and David MacKenzie.
Copyright (C) 2004 Free Software Foundation, Inc.
This is free software; see the source for copying
➥conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS
➥FOR A PARTICULAR PURPOSE.
```

Uzyskiwanie informacji o ilości wolnego miejsca na partycjach

Aby sprawdzić, ile miejsca pozostało jeszcze na partycjach założonych w systemie i ile miejsca zajmują znajdujące się na nich obecnie elementy, używany polecenia `df`.

```
[lukasz@linux linux]$ df
System plików         bl.  1K B        użyte dostępne %uż.
➥zamont. na
/dev/hda3              3281116   2802636    311808  90% /
```

```
/dev/hda5              350021    276540     55410  84%
↪/home
none                    62996         0     62996   0%
↪/dev/shm
```

A oto parametry, których można użyć z tym poleceniem:

`-m, --megabytes`
: Wyświetla informację o stopniu zajętości partycji dyskowej oraz o ilości wolnego miejsca na partycjach w bardziej przyswajalny sposób, podając wyniki w MB.

```
[lukasz@linux linux]$ df -m
System plików         bl.  1M B        użyte dostępne %uż.
↪zamont. na
/dev/hda3                3205      2737       305  90% /
/dev/hda5                 342       271        55  84%
↪/home
none                       62         0        62   0%
↪/dev/shm
```

Ustalanie, ile miejsca zajmuje plik lub katalog

Kiedy chcemy się zorientować, ile miejsca na dysku zajmuje dany plik czy katalog, możemy skorzystać z polecenia `du`. W przypadku plików wystarczy co prawda użyć polecenia `ls` z parametrem `-l`, dzięki któremu otrzymujemy przybliżony rozmiar elementów; jeżeli jednak chodzi o katalogi, sprawa nie jest już taka prosta. Z pomocą przychodzi nam wtedy wspomniane polecenie `du`. Podawana w wyniku jego wykonania ilość miejsca jest prezentowana w jednostkach dyskowych, jakimi są bloki.

```
[lukasz@linux linux]$ du plik1
1       plik1
2
```

A oto parametry, z których można skorzystać:

`-a, --all`
: Wyświetla ilość miejsca zajętego nie tylko przez sam katalog, ale także przez jego elementy.

```
[lukasz@linux linux]$ du kat2
1       kat2/kat3
2       kat2
```

`-b, --bytes`

Wyświetla informacje w bardziej przyswajalny sposób, w postaci nie bloków, ale liczby bajtów, którą dany element zajmuje na dysku.

```
[lukasz@linux linux]$ du -b kat2
2048    kat2
[lukasz@linux linux]$ du -a -b plik1
1024    plik1
```

`-c, --total`

Podaje sumę rozmiaru wszystkich elementów.

```
[lukasz@linux linux]$ du -c kat2
1       kat2/kat3
2       kat2
2       razem

[lukasz@linux linux]$ du -c -b kat2
1024    kat2/kat3
2048    kat2
2048    razem
```

`-S, --separate-dirs`

Pokazuje rozmiar każdego katalogu z osobna i nie włącza do tego rozmia2rów znajdujących się w nich podkatalogów.

```
[lukasz@linux linux]$ du -S -b -c kat2
1024    kat2/kat3
1024    kat2
2048    razem
```

`-s, --summarize`

Wypisuje całkowitą objętość danego elementu bez podawania zbędnych informacji.

```
[lukasz@linux linux]$ du -s -b kat2
2048    kat2
```

Polecenia more i less

Polecenia `more` i `less` będą pomocne, kiedy zechcemy przeczytać plik lub kiedy rezultat wykonania jakiegoś polecenia nie zmieści się na ekranie.

Polecenie `more` pozwala nam poruszać się tylko w dół pliku. Polecenia tego możemy użyć oddzielnie, niezależnie od innych, lub wpisując je po znaku potoku, aby przekazać w ten sposób na jego wejście dane z innego polecenia.

```
[lukasz@linux linux]$ more plik1
aaaaaa
...
...
...
aaaaaaa
--More--(83%)
```

Przekazanie wyników innego polecenia do `more` jest bardzo przydatne, jeżeli wynik wyświetlania zawartości danego katalogu nie mieści się na ekranie. Jak wspomniałem, osiągniemy to, używając znaku potoku (`|`).

```
[lukasz@linux linux]$ ls -l /etc | more
-rw-r--r--    1 root     root        15228 sty 24  2003
↪a2ps.cfg
-rw-r--r--    1 root     root         2562 sty 24  2003
↪a2ps-site.cfg
[...]
[...]
[...]
-rw-r--r--    1 root     root           47 lut 21 15:00
↪adjtime
drwxr-xr-x    2 root     root         4096 lip  4  2003 aep
--More--
```

Polecenie `less` jest podobne do `more`, ale umożliwia poruszanie się w przód i w tył pliku lub wyniku polecenia. Także w tym przypadku możemy wpisać polecenie osobno lub za znakiem potoku. Po pliku poruszamy się, używając klawiszy strzałek.

```
[lukasz@linux linux]$ less plik1
aaaaaa
...
...
...
aaaaaaa
plik1
```

Czyszczenie terminala

Okno terminala możemy wyczyścić tak, aby znak zachęty znajdował się na jego górze; dzięki temu będziemy mieli więcej miejsca, co pozwoli nam na wykonanie nowych operacji i lepszą orientację w zawartości terminala. Terminal zostanie wyczyszczony po wpisaniu polecenia `clear`:

```
[lukasz@linux linux]$ clear
```

Montowanie i odmontowywanie systemów plików

W systemie Linux partycje i napędy możemy swobodnie montować i odmontowywać. Zamontowania przed użyciem wymagają takie nośniki jak dyskietka czy CD-ROM; po użyciu trzeba z kolei je odmontować. Jeśli nie przeprowadzimy procesu montowania, nie będziemy mogli uzyskać dostępu do danych znajdujących się na tych nośnikach. Do montowania napędów służy polecenie `mount`. Odpowiednie partycje mają określone numery, które są potrzebne dla określenia ich nazwy. Z napędami dyskietek i płyt CD jest trochę inaczej. Domyślnie napędy te mają własne, przypisane im nazwy. Dla przykładu napędy dyskietek to *fd0* i *fd1*; oba są montowane w katalogu */mnt*. Najlepiej objaśnić to na przykładzie.

Zamontujemy teraz napęd dyskietek. Przed zamontowaniem dyskietki katalog z zawartością napędu jest pusty, ale po zamontowaniu znajdują się w nim dane zapisane na dyskietce. Montowania

dokonujemy, podając nazwę napędu lub partycji, którą mamy zamiar zamontować:

```
[lukasz@linux lukasz]$ ls -l /mnt/floppy
razem 0
[lukasz@linux lukasz]$ mount /dev/fd0
[lukasz@linux lukasz]$ ls -l /mnt/floppy
razem 940
-rwxr-xr-x    1 lukasz   lukasz      35905 gru  4 22:27
↪bip.tar.gz
```

Z poleceniem `mount` możemy zastosować kilka opcji; oto one:

`-V`

Parametr ten informuje, jaką wersję oprogramowania `mount` mamy zainstalowaną.

```
[lukasz@linux lukasz]$ mount -V
mount: mount-2.11y
```

`-n`

Parametr ten pozwala na montowanie systemu plików bez zapisywania o tym informacji w pliku */etc/mtab*.

`-r`

Umożliwia montowanie systemu plików w trybie tylko do odczytu jego zawartości.

`-w`

Użycie tej opcji powoduje zamontowanie systemu plików w trybie do odczytu i zapisu.

System plików możemy także (oprócz umieszczenia go w domyślnym miejscu montowania) zamontować w naszym własnym katalogu.

W swoim katalogu głównym utworzyłem katalog o nazwie *dyskietka*; teraz zamontuję doń zawartość stacji dyskietek. Pierwszym podanym parametrem jest nazwa urządzenia lub partycji, a drugim miejsce, w którym ma być dostępna jego (lub jej) zawartość.

```
[root@linux lukasz]# mount /dev/fd0
↪/home/lukasz/linux/dyskietka
[root@linux lukasz]# ls -l /mnt/floppy
razem 0
[root@linux lukasz]# ls -l /home/lukasz/linux/dyskietka
razem 940
-rwxr-xr-x    1 root     root        35905 gru  4 22:27
↪bip.tar.gz
```

Kiedy już skończyliśmy korzystać z zamontowanego systemu plików, powinniśmy go odmontować, aby móc wymienić go na inny (w przypadku dyskietki lub płyty CD) albo chronić go przed niepożądanym zapisem. Używamy w tym celu polecenia `umount` wraz z nazwą urządzenia:

```
[lukasz@linux lukasz]$ umount /dev/fd0
[lukasz@linux lukasz]$ ls -l /mnt/floppy
razem 0
```

`-r`

Użycie tego parametru spowoduje, że jeżeli podczas odmontowania systemu plików pojawią się błędy, program spróbuje zamontować ponownie ten sam system w trybie tylko do odczytu.

Aktualna ścieżka, pod którą pracujemy

Do wyświetlenia aktualnej ścieżki stosuje się polecenie `pwd`. Wyświetla ono całą ścieżkę (od katalogu głównego na dysku), w której obecnie się znajdujemy.

```
[lukasz@localhost ~]$ pwd
/home/lukasz
```

Przełączanie się na konto innego użytkownika

Możemy przełączyć się na konto innego użytkownika w systemie, jeżeli posiadamy uprawnienia administratora lub jeżeli znamy hasło tego użytkownika. Jest to bardzo przydatne, jeżeli pracujemy

na serwerze i w pewnym momencie potrzebujemy dostać się szybko na konto użytkownika *root*, dzięki któremu będziemy mogli wykonać pewne polecenia, do których normalnie (korzystając z konta zwykłego użytkownika) nie mamy uprawnień. Przełączanie się na inne konto jest możliwe po wydaniu polecenia `su`.

```
[lukasz@linux lukasz]$ su root
Password:
```

Po poleceniu `su` wpisujemy `login` użytkownika, na którego konto chcemy się przełączyć. Potem system pyta nas o hasło tego użytkownika; oczywiście musimy je podać.

Uzyskiwanie informacji o sprzęcie

Aby dowiedzieć się więcej o komputerze, na którym pracuje nasz system, można posłużyć się dwoma poleceniami: `arch` i `uname`. Oba służą do wypisywania informacji o typie sprzętu komputerowego. Polecenia te są bardzo przydatne, kiedy wybieramy dystrybucję systemu lub gdy chcemy zainstalować dodatkowy program i musimy najpierw zorientować się, jakiego pakietu potrzebujemy i czy program ten będzie działał w naszym systemie.

Polecenie arch

Polecenie to wyświetla informacje o architekturze komputera, czyli o rodzaju zastosowanego procesora.

```
[lukasz@linux lukasz]$ arch
i686
```

Możliwe wyniki wydania tego polecenia to informacja o zastosowaniu procesora zgodnego z Intelem lub innego:

- „i386", „i486", „i586" itd.,
- „alpha", „sparc", „arm", „m68k", „mips", „ppc".

Polecenie uname

Polecenie to jest bardziej rozbudowane niż samo `arch`. Posiada kilka parametrów, dzięki którym możemy nie tylko dowiedzieć się, na jakim sprzęcie komputerowym pracujemy, ale także jakiej wersji oprogramowania używamy.

Wydanie polecenia `uname` bez parametru spowoduje wyświetlenie tylko informacji o używanym systemie operacyjnym.

```
[lukasz@linux lukasz]$ uname
Linux
```

`-a, --all`
: Po użyciu tego parametru zostaną wypisane wszystkie informacje o systemie.

```
[lukasz@linux lukasz]$ uname -a
Linux linux 2.4.20-9 #1 Wed Apr 2 13:42:50 EST 2003
↪i686 athlon i386 GNU/Linux
```

`-m, --machine`
: Parametr `-m` zastosowany w poleceniu `uname` daje rezultat taki sam jak polecenie `arch` — wyświetla architekturę maszyny.

```
[lukasz@linux lukasz]$ uname -m
i686
```

`-n, --nodename`
: Parametr ten wypisuje nazwę *hosta* naszego komputera w sieci.

```
[lukasz@linux lukasz]$ uname -n
linux
```

`-p, --processor`
: Wypisuje typ procesora w naszym komputerze.

```
[lukasz@linux lukasz]$ uname -p
athlon
```

`-r, --release`
Parametr ten pozwala uzyskać informacje na temat wersji jądra Linuksa.

```
[lukasz@linux lukasz]$ uname -r
2.4.20-9
```

`-s, --sysname`
Wypisuje nazwę systemu operacyjnego zainstalowanego na naszym komputerze.

```
[lukasz@linux lukasz]$ uname -s
Linux
```

`-v`
Wypisuje wersję systemu operacyjnego zainstalowanego na naszym komputerze.

```
[lukasz@linux lukasz]$ uname -v
#1 Wed Apr 2 13:42:50 EST 2003
```

Informacje o użytkowniku

Jeżeli potrzebujemy informacji o określonym użytkowniku, możemy użyć polecenia `finger` wraz z nazwą konta tego użytkownika:

```
[root@linux sbin]# finger lukasz
Login: lukasz                           Name: Lukasz Sosna
Directory: /home/lukasz                 Shell: /bin/bash
On since Sat Feb 21 18:45 (CET) on :0 (messages off)
On since Sat Feb 21 18:45 (CET) on pts/0   26 minutes
↪25 seconds idle
On since Sat Feb 21 19:07 (CET) on pts/1 (messages off)
No mail.
No Plan.
```

W rezultacie otrzymujemy informacje o powłoce, której używa, miejscu, które zostało mu przydzielone na dysku, oraz o czasie ostatniego logowania.

Kto jest obecnie zalogowany

Jeżeli chcemy wiedzieć, jacy użytkownicy są obecnie zalogowani do systemu, wystarczy, że użyjemy niepozornego polecenia `w`:

```
[root@linux sbin]# w
 19:15:05  up 31 min,  1 users,  load average: 0.14, 0.10, 0.10
USER     TTY      FROM              LOGIN@   IDLE   JCPU
↪PCPU  WHAT
lukasz   :0       -                 6:45pm   ?     0.00s
↪0.07s  /bin/bash
```

Po wydaniu tego polecenia możemy także zorientować się, co dany użytkownik robi i ile procent zasobów komputera obecnie zużywa.

Informacje o pamięci systemowej

Aby dowiedzieć się, ile pamięci systemowej *RAM* i *Swap* jest obecnie zajęte, wydajemy polecenie `free`:

```
[lukasz@linux lukasz]$ free
             total       used       free     shared
↪buffers     cached
Mem:        125992     121004       4988          0
↪3508      55556
-/+ buffers/cache:      61940      64052
Swap:       257000      20648     236352
```

Omawiane polecenie możemy użyć z parametrami:

`-jednostka`
: Parametr ten pozwala nam na wybranie jednostki miary, w jakiej będzie wyświetlana wielkość poszczególnych pamięci:

  - `-b` — wyświetla informacje w bajtach;
  - `-k` — wyświetla informacje w kilobajtach;
  - `-m` — wyświetla informacje w megabajtach.

```
[lukasz@linux lukasz]$ free -b
             total       used       free     shared
↪buffers     cached
Mem:     129015808  123994112    5021696          0
↪3588096   56893440
-/+ buffers/cache:   63512576   65503232
Swap:    263168000   21180416  241987584
```

Przeglądanie kalendarza

Polecenie `cal` wyświetla prosty kalendarz w formacie tekstowym. W przypadku, gdy nie podamy argumentów, wyświetlany jest bieżący miesiąc.

```
[lukasz@localhost ~]$ cal
      maj 2006
ni po wt śr cz pi so
    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
```

`-m`

W przypadku, gdybyśmy chcieli wyświetlić poniedziałek jako pierwszy dzień tygodnia, należy użyć dodatkowo tego parametru. W przypadku braku jakiegokolwiek parametru domyślny dzień początku tygodnia to niedziela.

```
[lukasz@localhost ~]$ cal -m
      maj 2006
po wt śr cz pi so ni
 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
```

`-j`

Za pomocą tego narzędzia możemy także dowiedzieć się, który dzień z kolei mamy. Istnieje specjalna opcja, która wyświetli liczbę dni od dnia pierwszego stycznia bieżącego roku.

```
[lukasz@localhost ~]$ cal -j
          maj 2006
nie pon wto śro czw pią sob
    121 122 123 124 125 126
127 128 129 130 131 132 133
134 135 136 137 138 139 140
141 142 143 144 145 146 147
148 149 150 151
```

`-y`

Za pomocą tego narzędzia możesz wyświetlić również cały kalendarz na bieżący rok. Dzięki temu będziesz miał wgląd do wszystkich dni w roku.

```
[lukasz@localhost ~]$ cal -y
[...]
      kwiecień                maj                 czerwiec
ni po wt śr cz pi so   ni po wt śr cz pi so   ni po wt śr cz pi so
                   1       1  2  3  4  5  6                 1  2  3
 2  3  4  5  6  7  8    7  8  9 10 11 12 13    4  5  6  7  8  9 10
 9 10 11 12 13 14 15   14 15 16 17 18 19 20   11 12 13 14 15 16 17
16 17 18 19 20 21 22   21 22 23 24 25 26 27   18 19 20 21 22 23 24
23 24 25 26 27 28 29   28 29 30 31            25 26 27 28 29 30
30
[...]
```

Wyświetlanie dowolnych lat w formie kalendarza w konsoli jest także możliwe. Po poleceniu `cal` należy podać pełną czterocyfrową notację roku, aby móc go zobaczyć.

```
[lukasz@localhost ~]$ cal 2003
[...]
      kwiecień                maj                 czerwiec
ni po wt śr cz pi so   ni po wt śr cz pi so   ni po wt śr cz pi so
       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
[...]
```

Aktualizacja daty i czasu

Polecenie służy do wyświetlania i ustawiania w systemie odpowiedniej daty i czasu systemowego. Dzięki temu możesz zmienić datę i czas podczas pracy za pomocą konsoli.

```
[lukasz@localhost ~]$ date
śro maj 24 17:49:40 CEST 2006
```

`%H`

W celu wyświetlenia tylko obecnej godziny w formacie 24-godzinnym należy podać do polecenia odpowiedni parametr.

```
[lukasz@localhost ~]$ date +%H
17
```

`%I`

Parametr ten powoduje wyświetlenie aktualnej godziny w formacie 12-godzinnym.

```
[lukasz@localhost ~]$ date +%I
05
```

`%M`

Parametr ten służy do wyświetlania aktualnej minuty, bez wyświetlania godziny.

```
[lukasz@localhost ~]$ date +%M
52
```

`%p`

Jeżeli wyświetlamy czas w postaci 12-godzinnej i chcemy zobaczyć, czy jest to czas przed czy po południu, powinniśmy użyć opcji `%p`, która wyświetli odpowiednią porę dnia.

```
[lukasz@localhost ~]$ date +%p
PM
```

`%r`

Parametru używamy w celu wyświetlenia czasu w formacie 12-godzinnym.

```
[lukasz@localhost ~]$ date +%r
05:54:12
```

`%s`

Za pomocą tego narzędzia możemy także pokazać, ile sekund minęło od *1 stycznia 1970* roku godziny *00:00:00* czasu UTC.

```
[lukasz@localhost ~]$ date +%s
1148486129
```

`%S`

Parametr daje nam także możliwość wyświetlania liczby sekund, które upłynęły od pełnej minuty.

```
[lukasz@localhost ~]$ date +%S
25
```

`%T`

Parametr służy do wyświetlania pełnego czasu godzin wraz z minutami i sekundami. W tym przypadku czas zostanie wyświetlony w formacie 24-godzinnym.

```
[lukasz@localhost ~]$ date +%T
17:58:37
```

`+%X`

Dzięki narzędziu możemy dowiedzieć się, jaka jest lokalna reprezentacja czasu na naszym komputerze.

```
[lukasz@localhost ~]$ date +%X
17:58:56
```

`%b`

W celu zobaczenia skróconej nazwy obecnego miesiąca należy za poleceniem wpisać parametr `%b`. Pojawi się wówczas trzyliterowa nazwa obecnego miesiąca.

```
[lukasz@localhost ~]$ date +%b
Maj
```

`%B`

Za pomocą polecenia możemy także zobaczyć pełną lokalną nazwę miesiąca.

```
[lukasz@localhost ~]$ date +%B
maj
```

`%c`

Parametr ten służy do wyświetlenia lokalnej daty i czasu w miejscu, w którym się znajdujemy. Dzięki temu data i czas zostaną wyświetlone w formacie, jaki został ustawiony w systemie.

```
[lukasz@localhost ~]$ date +%c
śro 24 maj 2006 18:03:19 CEST
```

`%d`

Ten parametr służy do wyświetlania aktualnego dnia miesiąca.

```
[lukasz@localhost ~]$ date +%d
24
```

`%D`

W celu wyświetlenia daty w formacie porozdzielanym znakami slash, należy użyć parametru `%D`. Pamiętaj o tym, że czas podany jest w formacie `miesiąc/dzień/rok`.

```
[lukasz@localhost ~]$ date +%D
05/24/06
```

`%j`

Parametr ten powoduje wyświetlenie obecnego dnia roku; dzięki niemu możemy dowiedzieć się, który to dzień z kolei.

```
[lukasz@localhost ~]$ date +%j
144
```

`%m`

Parametr ten służy do wyświetlania informacji o obecnym numerze miesiąca.

```
[lukasz@localhost ~]$ date +%m
05
```

`%U`

Parametr ten powoduje wyświetlenie numeru tygodnia w roku. Niedziela w tym wypadku jest uznawana jako pierwszy dzień tygodnia.

```
[lukasz@localhost ~]$ date +%U
21
```

%w

Za pomocą tego parametru wyświetlimy dzień tygodnia, gdzie 0 odpowiada niedzieli.

```
[lukasz@localhost ~]$ date +%w
3
```

%W

Parametr wyświetli numer tygodnia. Tydzień w wypadku użycia tego parametru rozpoczyna się od poniedziałku.

```
[lukasz@localhost ~]$ date +%W
21
```

%x

Parametru używamy w przypadku, gdy chcemy wyświetlić lokalną reprezentację daty w formacie obowiązującym u nas w kraju czyli RRRR-MM-DD.

```
[lukasz@localhost ~]$ date +%x
2006-05-24
```

%y

Jeżeli chcemy wyświetlić dwie ostatnie cyfry roku, musimy użyć tego parametru.

```
[lukasz@localhost ~]$ date +%y
06
```

%Y

Parametr podaje w wyniku pełny numer roku w notacji czterocyfrowej.

```
[lukasz@localhost ~]$ date +%Y
2006
```

Datę możemy ustawić za pomocą polecenia bez wywołania parametru +. W czasie tego wywołania data jest ustawiana na datę podaną po poleceniu, która musi składać się z następujących pól:

- `MM` — miesiąc,
- `DD` — dzień miesiąca,
- `hh` — godzina,
- `mm` — minuta,
- `CC` — pierwsze dwie cyfry roku (opcjonalne),
- `YY` — ostatnie dwie cyfry roku (opcjonalne).

Można także ustawić datę za pomocą parametru `-s`, który służy do definiowania daty w pełnym formacie. Należy wówczas podać pełną czterocyfrową notację roku, miesiąc, dzień, godzinę i minutę.

```
[lukasz@localhost ~]$ date -s '2006-01-01 12:36'
```

`-r`

W celu wyświetlenia daty ostatniej modyfikacji pliku możemy użyć polecenia `date` wraz z parametrem i nazwą pliku za tym parametrem.

```
[lukasz@localhost ~]$ date -r plik.txt
śro maj 24 17:30:39 CEST 2006
```

`--version`

Parametru używamy w celu wyświetlenia wersji programu (uzyskujemy pełne informacje o programie i jego licencji).

```
[lukasz@localhost ~]$ date --version
date (coreutils) 5.2.1
Napisany przez David MacKenzie.
Copyright (C) 2004 Free Software Foundation, Inc.
Ten program jest darmowy; warunki kopiowania są opisane
↪w źródłach.
Autorzy nie dają ŻADNYCH gwarancji, w tym również
↪gwarancji PRZYDATNOŚCI
DO SPRZEDAŻY LUB DO KONKRETNYCH CELÓW.
```

Kontrolowanie wysyłania wiadomości

Do wysyłania wiadomości służy polecenie `write`, jednak z wyłączoną możliwością wysyłania komunikatów nie będziesz mógł za pomocą konsoli porozumiewać się z innymi użytkownikami — właśnie dlatego należy włączyć taką możliwość.

`y`

W celu włączenia dostępu do swojego terminala należy podać polecenie `mesg` właśnie z tym atrybutem.

```
[lukasz@localhost ~]$ mesg y
```

`n`

W celu wyłączenia dostępu do swojego terminala wpisz polecenie `mesg` z atrybutem `n`.

```
[lukasz@localhost ~]$ mesg n
```

Wysyłanie wiadomości do innego użytkownika

Za pomocą polecenia `write` możesz wysłać dowolny komunikat tekstowy do innego użytkownika w swoim systemie.

```
[lukasz@localhost ~]$ write [użytkownik] [nazwatty] wiadomość
```

Pole `[użytkownik]` to nazwa użytkownika, do którego masz zamiar wysłać wiadomość; pole `[nazwatty]` to nazwa, pod jaką pracuje użytkownik.

```
[lukasz@localhost ~]$ write lukasz informacja
```

Użytkownik lukasz otrzyma informację o tym, że wysłałeś do niego wiadomość.

```
Message from lukasz@localhost on tty1 at 20:18 informacja
```

Wysyłanie wiadomości z pliku tekstowego

Wiadomości z pliku tekstowego wysyłamy za pomocą polecenia `wall`, po którym podajemy plik tekstowy z naszą wiadomością.

```
wall plik.txt
Broadcast message from lukasz (pts/5) (Mon Jun 5 14:02:20 2006):
plik.txt
```

Wiadomość zostanie rozesłana do wszystkich użytkowników w sieci.

Wysyłanie komunikatów do wszystkich sieci z pliku tekstowego

Wiadomość zapisaną w pliku tekstowym można wysłać do wszystkich użytkowników sieci.

```
rwall plik.txt
Broadcast message from lukasz (pts/5) (Mon Jun 5 14:02:20 2006):
plik.txt
```

Pokazywanie ostatnio zalogowanych użytkowników

Za pomocą polecenia `last` możemy dowiedzieć się wielu informacji na temat tego, kto korzystał z komputera pracującego pod kontrolą systemu Linux, a także sprawdzić, kiedy to było.

```
[lukasz@localhost ~]$last
apache2  :0                              Thu Jan  5
↪10:40 - 10:40  (00:00)
reboot   system boot  2.6.12-12mdk     Thu Jan  5
↪10:39 (00:28)
lukasz   pts/1                           Thu Jan  5
↪00:34 - 00:35  (00:01)
lukasz   :0                              Thu Jan  5
↪00:34 - 00:35  (00:01)
```

```
lukasz   :0                              Thu Jan  5
↪00:34 - 00:34  (00:00)
root     pts/1                           Thu Jan  5
↪00:32 - 00:34  (00:02)
root     :0                              Thu Jan  5
↪00:32 - 00:34  (00:02)
root     :0                              Thu Jan  5
↪00:32 - 00:32  (00:00)
lukasz   pts/2                           Thu Jan  5
↪00:31 - 00:31  (00:00)
lukasz   pts/1                           Thu Jan  5
↪00:29 - 00:32  (00:02)
```

`-n`

Parametr `-n` decyduje o tym, ile wierszy wyników ma zostać pokazanych po wywołaniu tego polecenia.

```
[lukasz@localhost ~]$ last -n 15
apache2  pts/2                           Wed May 24
↪17:05   still logged in
apache2  pts/1                           Wed May 24
↪17:03   still logged in
apache2  :0                              Wed May 24
↪17:03   gone - no logout
apache2  :0                              Wed May 24
↪17:03 - 17:03  (00:00)
reboot   system boot  2.6.12-12mdk       Wed May 24
↪17:02          (02:07)
reboot   system boot  2.6.12-12mdk       Wed May 24
↪17:01          (00:00)
apache2  pts/1                           Wed May 24
↪11:39 - down   (01:55)
```

`-R`

Parametru używa się w celu sprawdzenia, kto korzystał z naszego komputera, jednak bez wyświetlania numeru hosta, na którym pracował.

```
[lukasz@localhost ~]$ last -R
lukasz   :0           Thu Nov 24 15:41 - 15:41  (00:00)
reboot   system boot  Thu Nov 24 15:41          (00:02)
lukasz   pts/1        Thu Nov 24 15:38 - 15:39  (00:00)
lukasz   pts/0        Thu Nov 24 15:29 - 15:40  (00:10)
lukasz   :0           Thu Nov 24 15:28 - 15:40  (00:11)
lukasz   :0           Thu Nov 24 15:28 - 15:28  (00:00)
reboot   system boot  Thu Nov 24 15:28          (00:11)
```

`-a`

Jeżeli chcemy zobaczyć nazwę hosta, musimy wpisać ten parametr, wówczas w ostatniej kolumnie pojawi się jego nazwa.

```
[lukasz@localhost ~]$ last -a
reboot   system boot  Thu Nov 24 15:41          (00:02)
↪2.6.12-12mdk
lukasz   pts/1        Thu Nov 24 15:38 - 15:39  (00:00)
lukasz   pts/0        Thu Nov 24 15:29 - 15:40  (00:10)
lukasz   :0           Thu Nov 24 15:28 - 15:40  (00:11)
lukasz   :0           Thu Nov 24 15:28 - 15:28  (00:00)
reboot   system boot  Thu Nov 24 15:28          (00:11)
↪2.6.12-12mdk
```

`-d`

Parametru używamy w celu wyświetlenia logowań nielokalnych. Dzięki temu uzyskamy adres IP hosta, z którego nastąpiło logowanie.

```
[lukasz@localhost ~]$ last -d
reboot   system boot  0.0.0.0          Thu Feb  9
↪09:47          (00:01)
reboot   system boot  0.0.0.0          Thu Feb  9
↪09:46          (00:00)
apache2  pts/1        0.0.0.0          Mon Feb  6
↪12:22 - down   (01:01)
apache2  :0           0.0.0.0          Mon Feb  6
↪12:21 - 13:23  (01:01)
apache2  :0           0.0.0.0          Mon Feb  6
↪12:21 - 12:21  (00:00)
```

`-x`

Parametr powoduje wyświetlenie informacji o zmianach trybu pracy systemu i jego zamknięciu.

```
[lukasz@localhost ~]$ last -x
lukasz   pts/1                          Thu Nov 24
↪15:38 - 15:39  (00:00)
lukasz   pts/0                          Thu Nov 24
↪15:29 - 15:40  (00:10)
lukasz   :0                             Thu Nov 24
↪15:28 - 15:40  (00:11)
lukasz   :0                             Thu Nov 24
↪15:28 - 15:28  (00:00)
```

```
runlevel (to lvl 5)   2.6.12-12mdk     Thu Nov 24
↪15:28 - 15:40  (00:11)
reboot   system boot  2.6.12-12mdk     Thu Nov 24
↪15:28          (00:11)
```

Sprawdzanie, kto jest aktualnie zalogowany na naszym komputerze

Polecenie `who` wypisze wszystkich użytkowników zalogowanych do naszego systemu, którzy są obecnie dostępni.

```
[lukasz@localhost ~]$ who
lukasz pts/1 Jun 5 12:35
```

Informację o naszym loginie uzyskujemy za pomocą parametru `am i`.

```
[lukasz @localhost ~]$ who am i
lukasz pts/1 Jun 5 12:35
```

Informacja o tym, kto jest zalogowany do systemu

W celu zobaczenia użytkowników, którzy są zalogowani do systemu, należy użyć polecenia `users`, które wypisuje listę użytkowników w systemie.

```
[lukasz@localhost ~]$ users
lukasz
```

Sprawdzanie swojej nazwy użytkownika

W celu sprawdzenia swojej nazwy użytkownika korzystamy z polecenia `whoami` bez podawania żadnych parametrów.

```
[lukasz@localhost ~]$ whoami
lukasz
```

Pokazywanie lub ustawianie nazwy hosta systemowego

Do pokazywania lub ustawiania nazwy hosta systemowego służy polecenie `hostname`, dzięki któremu możemy pokazać lub zmienić nazwę hosta, domeny lub węzła systemu.

Po wpisaniu samego polecenia (bez żadnych parametrów) uzyskamy nazwę hosta, na którym obecnie pracujemy.

```
[lukasz@localhost ~]$ hostname
localhost
```

`-v, --verbose`
: Parametr wypisze informacje o tym, co w tej chwili dzieje się z naszą nazwą hosta.

```
[lukasz@localhost ~]$ hostname -v
gethostname()=`localhost'
localhost
```

`-i, --ip-address`
: Parametru używa się w celu wyświetlenia adresu IP hosta, na którym pracujemy.

```
[lukasz@localhost ~]$ hostname -i
127.0.0.1
```

`-a`
: Parametr służy wypisaniu nazwy zastępczej aliasu komputera.

```
[lukasz@localhost ~]$ hostname -a
```

`-s`
: Dzięki temu parametrowi możemy wypisać skróconą nazwę komputera.

```
[lukasz@localhost ~]$ hostname -s
localhost
```

`-f`

Parametr ten pozwala na wypisanie pełnej nazwy komputera.

```
[lukasz@localhost ~]$ hostname -f
localhost
```

`-d`

Dzięki parametrowi uzyskujemy możliwość wypisywania nazwy domeny *DNS* komputera.

```
[lukasz@localhost ~]$ hostname -d
```

`-y`

Parametr pozwala na wypisanie nazwy domeny *NIS* komputera.

```
[lukasz@localhost ~]$ hostname -y
(none)
```

`-h`

Parametr pozwala na uzyskiwanie pomocy dotyczącej polecenia.

```
[lukasz@localhost ~]$ hostname -h
Usage: hostname [-v] {hostname|-F file}      set
➥hostname (from file)
       domainname [-v] {nisdomain|-F file}   set NIS
       ➥domainname (from file)
[...]
```

`-V, --version`

Ten parametr spowoduje wypisanie informacji o narzędziu i jego wersji.

```
[lukasz@localhost ~]$ hostname -V
net-tools 1.60
hostname 1.100 (2001-04-14)
```

Wyświetlanie i ustalanie parametrów interfejsu sieciowego

W celu wyświetlenia i ustalenia parametrów interfejsu sieciowego używamy polecenia `ifconfig`.

```
[lukasz@localhost ~]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:13:D4:64:57:C9
          inet6 addr: fe80::213:d4ff:fe64:57c9/64 Scope:Link
          [...]

eth1      Link encap:Ethernet  HWaddr 00:60:4C:4F:82:B2
          inet6 addr: fe80::260:4cff:fe4f:82b2/64 Scope:Link
          [...]

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

ppp0      Link encap:Point-to-Point Protocol
          inet addr:83.22.13.80  P-t-P:213.25.2.189
          ↪Mask:255.255.255.255
          [...]
```

`-a`

Parametru używamy w celu wyświetlenia informacji o wszystkich interfejsach na naszym komputerze.

```
[lukasz@localhost ~]$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:13:D4:64:57:C9
          inet6 addr: fe80::213:d4ff:fe64:57c9/64
          ↪Scope:Link
          [...]

eth1      Link encap:Ethernet  HWaddr 00:60:4C:4F:82:B2
          inet6 addr: fe80::260:4cff:fe4f:82b2/64
          ↪Scope:Link
          [...]

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]
```

```
ppp0      Link encap:Point-to-Point Protocol
          inet addr:83.22.13.80  P-t-P:213.25.2.189
          ↪Mask:255.255.255.255
         [...]

sit0      Link encap:IPv6-in-IPv4
          NOARP  MTU:1480  Metric:1
          [...]
```

Wyszukiwanie nazwy lub adresu IP zdalnego komputera

Polecenie `host` przez odpytywanie zdalnych serwerów *DNS* wyszukuje nazwę lub adres IP zdalnego komputera.

```
[lukasz@localhost ~]$ host www.helion.pl
www.helion.pl is an alias for virtual.helion.com.pl.
virtual.helion.com.pl has address 213.186.88.113
www.helion.pl is an alias for virtual.helion.com.pl.
www.helion.pl is an alias for virtual.helion.com.pl.
```

`-a`

Parametr, który pozwoli na uzyskanie wszystkich informacji o danym hoście (wskazanym jako parametr).

```
[lukasz@localhost ~]$ host -a www.helion.pl
Trying "www.helion.pl"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46046
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2,
↪ADDITIONAL: 2
;; QUESTION SECTION:
;www.helion.pl. IN ANY
[...]
Received 137 bytes from 194.204.152.34#53 in 41 ms
```

Sprawdzanie, czy dana domena jest już zarejestrowana

Sprawdzenia, czy dana domena została już zarejestrowana, dokonujemy poleceniem `whois`. Uzyskamy informację, czy domena została zarejestrowana, a jeżeli tak, to kto jest jej właścicielem.

```
[lukasz@localhost ~]$ whois www.helion.pl
Registrant:
[...]
```

Sprawdzenie dostępności hosta

Polecenie to wysyła do danego hosta niewielkie pakiety i oczekuje na odpowiedź od niego, którą następnie wyświetla na ekranie:

```
[lukasz@localhost ~]$ ping helion.pl
PING helion.pl (213.186.88.113) 56(84) bytes of data.
[...]
```

`-c N`

O tym, ile pakietów zostanie wysłanych, decydujemy za pomocą opcji `-c N`, gdzie `N` jest liczbą pakietów do wysłania.

```
[lukasz@localhost ~]$ ping -c 1 helion.pl
PING helion.pl (213.186.88.113) 56(84) bytes of data.

--- helion.pl ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time
↪2.26ms
```

`-i N`

Można także wysłać pakiety, decydując o tym, jaka będzie przerwa pomiędzy wysyłaniem kolejnych — służy do tego parametr `-i N`, gdzie `N` określa liczbę sekund pomiędzy wysyłaniem kolejnych pakietów.

```
[lukasz@localhost ~]$ ping -i 5 helion.pl
PING helion.pl (213.186.88.113) 56(84) bytes of data.
[...]
```

`-n`

Parametr służy do podania — zamiast nazwy komputera — jego adresu IP.

```
[lukasz@localhost ~]$ ping -n helion.pl
PING helion.pl (213.186.88.113) 56(84) bytes of data.
[...]
```

Czas, jaki upłynął od uruchomienia systemu

W celu wyświetlenia informacji o czasie, jaki upłynął od uruchomienia systemu, należy skorzystać z polecenia `uptime`.

```
[lukasz@localhost ~]$ uptime
12:59:11 up 2:27, 7 users, load average: 0.01, 0.05, 0.09
```

Polecenie to zwraca aktualny czas: *12:59:11*, czas działania systemu: *2:27*, informację o tym, ilu użytkowników jest obecnie do niego zalogowanych: 7, oraz średnie obciążenie komputera z ostatnich 15 minut.

Rozdział 3. Administrowanie systemem

W tym rozdziale omówię podstawowe funkcje administracyjne systemu, których na pewno każdy użytkownik Linuksa będzie używał nie raz. Rozdział ten nie zawiera opisu zarządzania siecią i oprogramowaniem, ponieważ każdy program do tego przeznaczony posiada wiele rozbudowanych opcji i na jego temat napisano już niejedną książkę; w tej pozycji natomiast na pewno nie wystarczyłoby miejsca na opisanie choćby jednego demona usług.

Poziom uruchomienia systemu

Linux może być uruchomiony na kilku poziomach pracy. Na każdym z tych poziomów podczas startu systemu uruchamiane są specjalne programy, które zostały zdefiniowane właśnie dla tego poziomu.

Te poziomy to:

- `0` — zatrzymanie systemu,
- `1` — tryb jednego użytkownika,
- `2` — tryb wieloużytkownikowy okrojony,
- `3` — tryb wieloużytkownikowy pełny,
- `4` — tryb pusty,
- `5` — tryb graficzny,
- `6` — restart systemu.

Poziomów `0` i `6` nie należy stosować ze zrozumiałych względów. Poziom pracy, na jakim uruchamia się system, jest zapisany w pliku: */etc/inittab*. Aby zmienić domyślny tryb uruchamiania komputera

pracującego pod kontrolą Linuksa, należy we wspomnianym pliku, w linii zawierającej następującą treść:

```
id:5:initdefault
```

wpisać wymagany poziom uruchamiania systemu. Najczęściej będą to poziomy `3` lub `5`.

Demony usług

Programy, które są zainstalowane w systemie, najczęściej posiadają swoje demony, dzięki którym mogą pracować cały czas. Kiedy system jest ponownie uruchamiany, taki demon automatycznie się ładuje oraz zatrzymuje podczas operacji zamknięcia systemu. Takie rozwiązanie nie wymaga od administratora ciągłego nadzorowania, czy dana usługa jest aktywna; po odpowiednim skonfigurowaniu systemu możemy być pewni, że wymagany demon zawsze się uruchomi.

Uruchamianie i zatrzymywanie

Usługi można na bieżąco uruchamiać, zatrzymywać i restartować. Każda z usług posiada te trzy możliwości, a niektóre dysponują także kilkoma innymi, specyficznych dla nich.

Każda usługa posiada własną nazwę; to właśnie nazwa pozwala zidentyfikować usługę. Usługi uruchamia się przez wywołanie odpowiedniego skryptu w katalogu */etc/init.d/*.

Dla przykładu uruchomimy serwer WWW *Apache*, który tutaj nosi nazwę `httpd`:

```
[root@linux root]# /etc/init.d/httpd start
Uruchamianie httpd:                                        [
OK   ]
```

Zatrzymanie demona usługi przebiega w analogiczny sposób:

```
[root@linux root]# /etc/init.d/httpd stop
Zatrzymywanie httpd:                                       [
OK   ]
```

Ustawianie demonów do startu w odpowiednim trybie

Demony — jak już wcześniej wspomniałem — potrafią samodzielnie uruchamiać się w odpowiednim trybie działania systemu. O trybach napisałem już wcześniej. Na pewno interesuje Cię, jak uruchomić danego demona w wybranym trybie pracy systemu. Wszystkie tryby posiadają swoją własną konfigurację, zapisaną w katalogu */etc/rc.d*.

W lokalizacji tej znajdują się katalogi o nazwach zgodnych z wzorcem *rcX.d*. W tym wypadku litera *X* określa poziom, na którym uruchamia się system. Jeżeli chcemy, aby do któregoś poziomu uruchamiania systemu została dodana funkcja automatycznego uruchamiania demona, musimy w odpowiednim katalogu stworzyć dowiązanie symboliczne do tego demona.

Dla przykładu załóżmy, że chcemy, aby w trybie `5` uruchamiał się automatycznie serwer WWW, którego nazwą demona jest *httpd*. Wystarczy stworzyć w katalogu */etc/rc.d/rc5d/* dowiązanie symboliczne do pliku tego demona, czyli do */etc/inid.d/httpd*.

Wyświetlenie informacji o działających usługach

Polecenie `ps` wyświetli nam informacje o usługach uruchomionych w naszym systemie. Dzięki niemu możemy w prosty sposób dowiedzieć się, jakie usługi działają, a jakie są wyłączone.

```
[lukasz@localhost ~]$ ps
PID TTY TIME CMD
7983 pts/5 00:00:00 bash
8839 pts/5 00:00:00 ps
```

`-U`

Parametr ten stosujemy w celu przeglądania własnych procesów, po parametrze należy podać nazwę użytkownika.

```
[lukasz@localhost ~]$ ps -U lukasz
PID TTY TIME CMD
5814 ? 00:00:00 startkde
5930 ? 00:00:00 dbus-launch
5931 ? 00:00:00 dbus-daemon-1
5939 ? 00:00:04 mdkapplet
[...]
```

`-c`

Parametr ten powoduje wypisanie wszystkich uruchomionych egzemplarzy danego procesu, którego nazwę podajemy za parametrem.

```
[lukasz@localhost ~]$ ps -c apache
```

`-p`

Parametru używamy w celu wypisania procesów o podanych numerach, po parametrze deklarujemy procesy o odpowiednich numerach; jeżeli chcemy mieć kilka procesów, oddzielamy je przecinkami.

```
[lukasz@localhost ~]$ ps -p1,2
PID TTY TIME CMD
1 ? 00:00:00 init
2 ? 00:00:00 ksoftirqd/0
```

Użytkownicy

Do dodawania nowego użytkownika do systemu służy polecenie `adduser`. Możemy w prosty sposób sprawdzić, ilu i jakich użytkowników obecnie zdefiniowano w systemie; wyświetlamy w tym celu zawartość katalogu */home*:

```
[root@linux root]# ls -l /home
razem 19
drwx------    2 root     root        12288 lip  4  2003
↪lost+found
drwxrwxrwx   40 lukasz   lukasz       4096 lut 21 19:22 lukasz
drwx------   35 tree     lukasz       2048 lis 20 21:59 tree
```

Konto nowego użytkownika dodajemy przez zdeklarowanie jego nazwy. Jest to najprostszy sposób:

```
[root@linux root]# adduser pingwin
```

Oczywiście użytkowników jest znacznie więcej niż ci, których konta sami zdefiniujemy za pomocą polecenia `adduser`. Na przykład istnieją także użytkownicy systemowi, którzy są przydzieleni do odpowiednich demonów usług, dzięki czemu usługi te działają z odpowiednimi, okrojonymi prawami dostępu.

A oto parametry, jakich można użyć z poleceniem `adduser`:

`-c komentarz`
: Do zakładanego konta doda komentarz. Będzie on umieszczony w pliku z hasłami.

`-e data_ważności`
: Określa datę ważności konta, po której stanie się ono nieaktywne. Datę zapisujemy w formacie `MM/DD/RR`.

  - `MM` — miesiąc,
  - `DD` — dzień,
  - `RR` — rok.

```
[root@linux root]# adduser -e 01/01/04 pingwin
```

`-f dni_nieaktywności`
: Parametr ten określa ilość dni po wygaśnięciu ważności hasła, jaka pozostała do wygaśnięcia ważności konta.

`-g grupa_początkowa`
: Parametr ten definiuje nazwę grupy początkowej dla danego użytkownika. Jeżeli grupa jeszcze nie istnieje, musimy ją utworzyć, ponieważ nazwa ta musi odnosić się do grupy, która już istnieje w systemie.

```
[root@linux root]# adduser -g apache pingwin
```

`-G grupa,[...]`

Ten parametr określa listę grup, do których będzie przypisany użytkownik. Kolejne grupy oddzielamy przecinkami.

```
[root@linux root]# adduser -G lukasz,apache pingwin
```

`-s powłoka`

Za pomocą tego parametru określamy powłokę dla użytkownika. Jeżeli pozostawimy pustą wartość, w roli powłoki użytkownika zostanie użyta ta, która jest ustawiona w systemie jako domyślna.

```
[root@linux root]# adduser -s /bin/sh pingwin
```

Konto użytkownika usuwamy z systemu za pomocą polecenia `userdel`. Oprócz samego użytkownika zostaje usunięty z systemu także jego katalog główny. Nie są jednak usuwane pliki pozostawione przez niego w innych lokalizacjach; jeśli chcemy je usunąć, musimy to zrobić sami.

```
[root@linux root]# userdel pingwin
```

`-r`

Polecenie `userdel` z parametrem `-r` usuwa katalog użytkownika w katalogu */home* i jego pliki znajdujące się w tym katalogu.

```
[root@linux root]# userdel -r pingwin
[root@linux root]# ls -l /home
razem 18
drwx------    2 root     root        12288 lip  4  2003
↪lost+found
drwxrwxrwx   40 lukasz   lukasz       4096 lut 21 19:22
↪lukasz
drwx------   35 tree     lukasz       2048 lis 20 21:59
↪tree
```

Grupy

Grupy to sposób na przydzielenie użytkownikowi dostępu do danego programu lub innego zasobu komputera. Nowe grupy dodaje się za pomocą polecenia `groupadd`, po którym należy podać nazwę dodawanej grupy.

```
[root@linux root]# groupadd grupa_pingwin
```

Usuwanie grup przeprowadzamy przez wykonanie analogicznego polecenia: `groupdel`; oczywiście także w tym przypadku musimy podać nazwę grupy — tej, która ma być usunięta.

```
[root@linux root]# groupdel grupa_pingwin
```

Nie możemy bezpośrednio usunąć grupy podstawowej użytkownika. Aby móc usunąć taką grupę, należy najpierw albo usunąć użytkownika z systemu, albo zmienić jego grupę początkową na inną.

Szukanie łańcuchów w bazie whatis

W celu znalezienia odpowiedniego łańcucha w bazie danych `whatis` możemy użyć polecenia `apropos`. Polecenie zwróci opis danego słowa kluczowego wpisanego po komendzie.

```
[lukasz@localhost ~]$ apropos config
```

Rozdział 4. Tworzenie skryptów powłoki

Pisanie skryptów powłoki to programowanie odpowiednich instrukcji — programów, które usprawniają wykonywanie wielu czynności. Skrypty powłoki obsługują zmienne, instrukcje warunkowe, pętle i wiele innych przydatnych elementów.

Skrypty powłoki to po prostu zgrupowane polecenia zapisane w jednym pliku. Podobnie jak przy wpisywaniu poleceń w okienku terminala, powinieneś pamiętać o tym, jak będą one wpisywane do pliku. Pamiętaj o tym, że każda nowa linia to nowe polecenie, więc nie można zapisywać polecenia w dwóch liniach.

Na przykład, aby wyświetlić listę zawartości swojego katalogu głównego, całe polecenie trzeba zapisać w jednej linii, ponieważ zapisanie go w dwóch lub więcej spowoduje błąd.

```
#!/bin/bash
vdir
/home/lukasz

[lukasz@localhost ~]$ ./skrypt
razem 44
drwx------ 5 lukasz lukasz 4096 gru 29 19:50 Desktop
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Dokumenty
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Muzyka
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Obrazy
-rw-r--r-- 1 lukasz lukasz 0 maj 24 13:04 pik.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik2.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik.txt
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Pobieranie
-rwxrwxrwx 1 lukasz lukasz 32 cze 10 13:41 skrypt
-rwxrwxrwx 1 lukasz lukasz 31 cze 10 13:41 skrypt~
drwx------ 6 lukasz lukasz 4096 maj 24 13:07 tmp
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Wideo
./skrypt: line 3: /home/lukasz: is a directory
```

Poprawnie zapisany skrypt będzie wyglądał następująco:

```
#!/bin/bash
vdir /home/lukasz
```

Wykonanie skryptu da pożądany efekt — listę zawartości katalogu głównego.

```
[lukasz@localhost ~]$ ./skrypt
razem 44
drwx------ 5 lukasz lukasz 4096 gru 29 19:50 Desktop
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Dokumenty
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Muzyka
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Obrazy
-rw-r--r-- 1 lukasz lukasz 0 maj 24 13:04 pik.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik2.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik.txt
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Pobieranie
-rwxrwxrwx 1 lukasz lukasz 31 cze 10 13:41 skrypt
-rwxrwxrwx 1 lukasz lukasz 29 cze 10 13:40 skrypt~
drwx------ 6 lukasz lukasz 4096 maj 24 13:07 tmp
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Wideo
```

Skrypty powłoki muszą zostać poprzedzone odpowiednią instrukcją odwołującą się do interpretera powłoki, której używamy.

```
#!/bin/bash
```

Dodatkowo plik taki musi mieć prawa do wykonywania, które należy nadać mu poleceniem `chmod` (opisywanym we wcześniejszej części książki).

```
[lukasz@localhost ~]$ chmod 777 skrypt
```

W celu uruchomienia skryptu należy odpowiednio go wywołać. Zapiszmy skrypt w pliku `skrypt`. Wówczas mamy go w katalogu głównym użytkownika i aby go wywołać, nie wystarczy wpisać jego nazwy, gdyż powłoka będzie wyszukiwała polecenia o takiej nazwie w ścieżkach wyszukiwania. Przed skryptem należy wpisać pełną ścieżkę dostępu do niego, zaczynając od znaku `/`, a gdy jesteśmy w katalogu, w którym jest umieszczony skrypt, wystarczy wpisać `./` (aktualny katalog, w którym znajduje się skrypt). W takim wypadku będziemy mieli pewność, że skrypt się uruchomi.

```
[lukasz@localhost ~]$ ./skrypt
```

Drugim sposobem uruchamiania skryptu jest użycie powłoki i przekazanie do niej skryptu w formie argumentu.

```
[lukasz@localhost ~]$ bash skrypt
razem 44
drwx------ 5 lukasz lukasz 4096 gru 29 19:50 Desktop
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Dokumenty
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Muzyka
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Obrazy
-rw-r--r-- 1 lukasz lukasz 0 maj 24 13:04 pik.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik2.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik.txt
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Pobieranie
-rwxrwxrwx 1 lukasz lukasz 31 cze 10 13:42 skrypt
-rwxrwxrwx 1 lukasz lukasz 32 cze 10 13:41 skrypt~
drwx------ 6 lukasz lukasz 4096 maj 24 13:07 tmp
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Wideo
```

Trzecim sposobem uruchomienia skryptu w aktualnej powłoce jest użycie znaku specjalnego. Dokonujemy tego za pomocą znaku `.`.

```
[lukasz@localhost ~]$ . skrypt
razem 44
drwx------ 5 lukasz lukasz 4096 gru 29 19:50 Desktop
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Dokumenty
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Muzyka
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Obrazy
-rw-r--r-- 1 lukasz lukasz 0 maj 24 13:04 pik.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik2.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik.txt
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Pobieranie
-rwxrwxrwx 1 lukasz lukasz 31 cze 10 13:42 skrypt
-rwxrwxrwx 1 lukasz lukasz 32 cze 10 13:41 skrypt~
drwx------ 6 lukasz lukasz 4096 maj 24 13:07 tmp
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Wideo
```

Zmienne

Zmienne to elementy, które mogą przechowywać wartości. W powłoce istnieją zmienne mogące przechowywać wartości logiczne, tekst i liczby. Nie trzeba deklarować typu zmiennej na samym początku skryptu — wystarczy podać dla niej wartość podczas wpisywania skryptu.

Zmienną definiuje przypisywana do niej wartość. Wartość do zmiennej najlepiej wpisywać w cudzysłowach (przy późniejszych manipulacjach jej wartością lub próbach użycia w innym miejscu skryptu cudzysłów zabezpiecza nas przed wystąpieniem błędu).

Zadeklarujmy zmienną nazywającą się `zmienna` i zawierającą słowo `tekst`.

```
#!/bin/bash
zmienna="tekst"
```

Jak widać, zmienna jest zwykłym tekstem. Przy jej deklarowaniu nie trzeba dodawać żadnych znaków specjalnych przed czy za nią.

```
#!/bin/bash
zmienna="tekst"
echo zmienna
```

Przy wyświetlaniu wartości zapisanej w zmiennej należy poprzedzić ją znakiem dolara „`$`", aby wyświetlanie zadziałało, to znaczy aby wyświetliła się jej wartość, a nie nazwa zmiennej.

```
#!/bin/bash
zmienna="tekst"
echo $zmienna
```

Gdybyśmy nie dodali znaku dolara przed nazwą zmiennej w instrukcji `echo`, po wywołaniu tego skryptu zostanie wyświetlona na ekranie wartość `tekst` zamiast wartości `zmienna`.

```
[lukasz@localhost ~]$ ./skrypt
zmienna
Skrypt bez dodania znaku dolara przed nazwą zmiennej

[lukasz@localhost ~]$ ./skrypt
tekst
Skrypt ze znakiem dolara przed nazwą zmiennej
```

Wypisywanie tekstu na ekranie użytkownika

Do wypisywania tekstu używamy kilku poleceń, spośród których najpopularniejszym jest `echo`.

W celu wypisania tekstu na ekranie użytkownika po poleceniu `echo` deklarujemy tekst, który zostanie wyświetlony po wywołaniu skryptu.

```
#!/bin/bash
echo To jest tekst
```

Po wywołaniu tego skryptu otrzymamy rezultat:

```
[lukasz@localhost ~]$ ./skrypt
To jest tekst
```

`-n`

Parametr ten nie wypisze na końcu linii znaku nowej linii, dzięki czemu wszystkie informacje zostaną wypisane w jednym wierszu.

```
#!/bin/bash
echo -n To jest tekst
echo To jest tekst

[lukasz@localhost ~]$ ./skrypt
To jest tekstTo jest tekst
```

`-e`

Parametr ten rozpoznaje i interpretuje wszystkie znaki specjalne wpisywane przez nas do skryptu. Znaki specjalne deklaruje się przez poprzedzenie ich znakiem backslasha.

```
#!/bin/bash
echo -e To jest tekst\a

[lukasz@localhost ~]$ ./skrypt
To jest tekst
```

`-E`

Parametr ten powoduje nieinterpretowanie znaków specjalnych we wpisywanym tekście i pomija ich wykonanie.

```
#!/bin/bash
echo -E To jest tekst\a

[lukasz@localhost ~]$ ./skrypt
To jest teksta
```

`\a`

Parametr powoduje pojawienie się alarmu w postaci sygnału dźwiękowego.

```
#!/bin/bash
echo -e To jest tekst\a
```

`\b`

Ten parametr po wypisaniu tekstu przesuwa kursor o jeden znak bliżej początku tekstu.

```
#!/bin/bash
echo -e To jest tekst\b
```

`\c`

Parametr powoduje niewypisanie znaku nowego wiersza na końcu linii.

```
#!/bin/bash
echo -e To jest tekst\c
```

`\f`

Ten parametr powoduje wysunięcie strony i zmianę miejsca kursora w tekście.

```
#!/bin/bash
echo -e To jest tekst\f
```

`\n`

Parametr powoduje pojawienie się nowego wiersza po zakończeniu wypisywania tekstu.

```
#!/bin/bash
echo -e To jest tekst\n
```

`\r`

Parametr powoduje powrót karetki do początku linii.

```
#!/bin/bash
echo -e To jest tekst\r
```

`\t`

Parametr powoduje pojawienie się znaku tabulacji w poziomie.

```
#!/bin/bash
echo -e To jest tekst\t
```

`\v`

Paremetr powoduje pojawienie się tabulacji w pionie.

```
#!/bin/bash
echo -e To jest tekst\v
```

`\\`

Parametr ten służy do wypisania znaku backslasha.

```
#!/bin/bash
echo -e To jest tekst\\
```

`\'`

Parametr ten pozwala na wypisanie pojedynczego cudzysłowu.

```
#!/bin/bash
echo -e To jest tekst\'
```

`\"`

Parametr pozwala na wypisanie podwójnego cudzysłowu.

```
#!/bin/bash
echo -e To jest tekst\"
```

`\nnn`

Parametr ten pozwala na wypisanie znaku z tabeli kodów ASCII o ósemkowej notacji.

```
#!/bin/bash
echo -e To jest tekst\nnn
```

Wartości logiczne

W powłoce — tak jak w każdym innym języku programowania — występują wartości logiczne, czyli wartości `TRUE` lub `FALSE`. W systemie wartość `0` zawsze oznacza prawdę, czyli `TRUE`, a jakakolwiek inna wartość oznacza fałsz, czyli wartość `FALSE`.

Wszystkie programy działające w powłoce zwracają informację o tym, czy udało im się poprawnie zakończyć działanie. Wartość ta jest umieszczana w specjalnej zmiennej `$?`.

```
#!/bin/bash
vdir /home/lukasz
echo $?
```

Program ten powinien na końcu wyświetlić liczbę określającą, czy udało mu się wyświetlić katalog, czy też nie.

```
[lukasz@localhost ~]$ ./skrypt
razem 44
drwx------  5 lukasz lukasz 4096 gru 29 19:50 Desktop
drwxrwxr-x  2 lukasz lukasz 4096 lis 24  2005 Dokumenty
drwxrwxr-x  2 lukasz lukasz 4096 lis 24  2005 Muzyka
drwxrwxr-x  2 lukasz lukasz 4096 lis 24  2005 Obrazy
-rw-r--r--  1 lukasz lukasz    0 maj 24 13:04 pik.txt
-rw-r--r--  1 lukasz lukasz    7 maj 24 17:30 plik2.txt
-rw-r--r--  1 lukasz lukasz    7 maj 24 17:30 plik.txt
drwxrwxr-x  2 lukasz lukasz 4096 lis 24  2005 Pobieranie
-rwxrwxrwx  1 lukasz lukasz   39 cze 11 18:30 skrypt
-rwxrwxrwx  1 lukasz lukasz   34 cze 11 18:20 skrypt~
drwx------  6 lukasz lukasz 4096 maj 24 13:07 tmp
drwxrwxr-x  2 lukasz lukasz 4096 lis 24  2005 Wideo
0
```

Jak widać powyżej, katalog został wyświetlony i dlatego program zwrócił wartość `TRUE`, czyli liczbę `0` na końcu kodu. W przypadku niepowodzenia zwróciłby wartość `1`, tak jak poniżej.

```
[lukasz@localhost ~]$ ./skrypt
vdir: /home/lukasz2: Nie ma takiego pliku ani katalogu
1
```

Polecenie test

Polecenie `test` służy do porównywania liczb lub ciągów znaków i wypisywania do zmiennej wartości porównania.

`-d`

Parametr sprawdza, czy plik o podanej nazwie jest katalogiem.

```
#!/bin/bash
test -d plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

`-f`

Parametr sprawdza, czy plik jest zwykłym plikiem, czy też z prawami do wykonywania.

```
#!/bin/bash
test -f plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

`-L`

Parametr sprawdza, czy plik jest dowiązaniem symbolicznym.

```
#!/bin/bash
test -L plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

`-r`

Parametr sprawdza, czy dany plik istnieje i czy można go odczytać.

```
#!/bin/bash
test -r plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

`-w`

Parametr sprawdza, czy dany plik istnieje i czy można go zapisać.

```
#!/bin/bash
test -w plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

`-x`

Parametr sprawdza, czy plik o danej nazwie istnieje i czy można go uruchomić.

```
#!/bin/bash
test -x plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

`-s`

Parametr sprawdza, czy dany plik został zapisany na dysku i czy jego wartość (długość) nie jest zerowa.

```
#!/bin/bash
test -s plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

`-nt`

Parametr sprawdza, czy *plik1* jest nowszy od *plik2*.

```
#!/bin/bash
test plik.txt -nt plik2.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

`-ot`

Parametr sprawdza, czy *plik1* jest starszy od *plik2*.

```
#!/bin/bash
test plik.txt -ot plik2.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

`=`

Parametr sprawdza, czy ciągi podane po jego obu stronach są identyczne.

```
#!/bin/bash
test "abc"="abc"
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

`!=`

Parametr sprawdza, czy ciągi podane po jego obu stronach nie są identyczne.

```
#!/bin/bash
test "abc"!="abc"
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

`-z`

Parametr sprawdza, czy ciąg podany za nim ma zerową długość.

```
#!/bin/bash
test -z abc
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

`-n`

Parametr sprawdza, czy ciąg podany za nim ma niezerową długość.

```
#!/bin/bash
test -n abc
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

`-eq`

Parametr sprawdza, czy wartości podane po jego obu stronach są sobie równe.

```
#!/bin/bash
test 1 -eq 1
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

`-ne`

Parametr sprawdza, czy wartości podane po jego obu stronach nie są sobie równe.

```
#!/bin/bash
test 1 -ne 1
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

`-gt`

Parametr sprawdza, czy pierwsza wartość jest większa od drugiej.

```
#!/bin/bash
test 2 -gt 1
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

`-ge`

Parametr sprawdza, czy pierwsza wartość jest większa lub równa drugiej.

```
#!/bin/bash
test 2 -ge 1
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

`-lt`

Parametr sprawdza, czy pierwsza wartość jest mniejsza od drugiej.

```
#!/bin/bash
test 2 -lt 1
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

`-le`

Parametr sprawdza, czy pierwsza wartość jest mniejsza lub równa drugiej.

```
#!/bin/bash
test 2 -le 1
echo $?

[lukasz@localhost ~]$ ./skrypt
1
!
```

Parametr ten służy do negowania testu, bardzo często stosuje się go w instrukcjach warunkowych i w pętlach.

Instrukcja if

Instrukcja sprawdza, czy dany warunek jest spełniony, i w zależności od tego wykonuje odpowiednie czynności.

```
if wartość
then
zrób coś
fi
Najprostszy wariant polecenia if

#!/bin/bash
if [ 1 = 1 ]
then
echo Wartosci sa rowne
fi
```

Przykład skryptu sprawdzającego, czy wartości są równe.

```
[lukasz@localhost ~]$ ./skrypt
Wartosci sa rowne
```

Ten warunek sprawdza, czy wartość jest spełniona, i w zależności od wyniku sprawdzenia wykonuje określoną operację (wartość spełniona — `zrób coś`, w przeciwnym wypadku — `zrób coś innego`).

```
if  wartość
then
zrób coś
else
zrób coś innego
fi

#!/bin/bash
if [ 1 = 2 ]
then
echo Wartosci sa rowne
else
echo Wartosci sa rozne
fi
```

Przykład skryptu sprawdzającego, czy wartości są równe, czy też nie.

```
[lukasz@localhost ~]$ ./skrypt
Wartosci sa rozne
```

Polecenie sprawdza, czy warunki są spełnione. W przypadku, gdyby którykolwiek z warunków został spełniony, wykona polecenie.

W przypadku, gdyby żaden z warunków nie został spełniony, wykona ono funkcję `zrób coś innego`.

```
If wartość
then
zrób coś
elif wartość2
then
zrób coś 2
elif ...
...
else
zrób coś innego
di
```

Instrukcja case

Instrukcja `case` sprawdza, czy warunek ma odpowiednią wartość, a następnie przechodzi do odpowiedniego fragmentu kodu w programie.

```
Case warunek in
odpoweidz1)
zrob coś 1
;;
Esac
```

Przykład skryptu sprawdzającego, jaką wartość ma liczba.

```
#!/bin/bash
wartosc=1
case "$wartosc" in
1)
echo Liczba ma wartosc 1
;;
2)
echo Liczba ma wartosc 2
;;
Esac

[lukasz@localhost ~]$ ./skrypt
Liczba ma wartosc 1
```

Pętla while

Pętla `while` powtarza wykonywanie określonych czynności dopóty, dopóki warunek w niej podany nie zostanie spełniony.

```
While polecenie
do
zrob coś
done
```

Oto skrypt z pętlą `while` wyświetlający liczbę i po każdym przejściu pętli (wykonaniu fragmentu skryptu) zwiększający ją o jeden. W przypadku, gdy liczba jest równa `2`, skrypt kończy działanie.

```
#!/bin/bash
i=0
while [ $i -lt 2 ]
do
echo $i
i=`expr $i + 1`
done

[lukasz@localhost ~]$ ./skrypt
0
1
```

Pętla until

Pętla `until` powtarza polecenie w niej zapisane dopóty, dopóki warunek nie zostanie spełniony.

```
Until polecenie
do
zrob coś
done
```

Pętla for

Pętla `for` powtarza daną czynność żądaną liczbę razy (określoną w podanej do niej liczbie wykonań).

```
For zmienna in lista
do
zrob coś
done
```

Pętla `for`, która wyświetla w kolejnych liniach wszystkie wymienione w niej owoce.

```
#!/bin/bash
for owoc in Jablko Pomarancza Cytryna
do
echo $owoc
done

[lukasz@localhost ~]$ ./skrypt
Jablko
Pomarancza
Cytryna
```

Break

Polecenie `break` kończy działanie pętli, jeżeli podamy warunek do jej zakończenia.

```
#!/bin/bash
for owoc in Jablko Pomarancza Cytryna
do
echo $owoc
if [ "$owoc" = "Pomarancza" ]
then
break
fi
done
```

Ten skrypt ma zakończyć działanie (za pomocą polecenia `break`) po pojawieniu się tekstu `Pomarancza`.

```
[lukasz@localhost ~]$ ./skrypt
Jablko
Pomarancza
```

Continue

Polecenie `continue` wymusza przejście do kolejnej iteracji, czyli następnego kroku.

```
#!/bin/bash
for owoc in Jablko Pomarancza Cytryna
do
echo $owoc
if [ "$owoc" = "Pomarancza" ]
then
continue
fi
echo tekst przerywnika
done
```

W napisanym przez nas kodzie zadaniem instrukcji `continue` jest ukrycie napisu `tekst przerywnika` podczas przetwarzania „owocu": `Pomarancza`.

```
[lukasz@localhost ~]$ ./skrypt
Jablko
tekst przerywnika
Pomarancza
Cytryna
tekst przerywnika
```

Argumenty pobierane z wiersza powłoki

Wszystkie napisane przez nas skrypty powłoki mogą przyjmować argumenty. Do argumentów wysłanych po uruchomieniu skryptu odwołujemy się za pomocą zmiennych `$1`, `$2`, `$3` ... `$n`.

```
#!/bin/bash
echo "Dzisiaj pogoda była $1"
```

Przykładowy skrypt pobierze pierwszą wartość za jego nazwą i wstawi ją w miejsce `$1`, a następnie wyświetli ją na ekranie.

```
[lukasz@localhost ~]$ ./skrypt super
Dzisiaj pogoda była super
```

Rozdział 5. Polecenia dodatkowe

SSH

SSH jest standardem protokołu używanego podczas komunikacji klient-serwer. Umożliwia on nawiązanie połączenia z serwerem za pomocą konsoli. Dzięki temu możemy wykonywać polecenia na zdalnym serwerze. Taka możliwość operacji wykonywanych na serwerze jest bardzo przydatna w czasach gwałtownego rozwoju internetu. Większość firm oferujących serwery dedykowane umożliwia realizację połączenia za pomocą tego protokołu i samodzielną administrację nim.

Jeśli korzystamy z systemu operacyjnego z rodziny Linux, możemy bez problemu wpisać polecenie `ssh` w konsoli. W przypadku komputera pracującego pod kontrolą systemu Windows trzeba pobrać odpowiedni do tego celu program *PuTTY*. Oprogramowanie jest darmowe, znajduje się na stronie: *http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html*.

`-1`
: Parametr ten wymusza połączenie z użyciem protokołu w wersji 1.

`-2`
: Parametr ten wymusza połączenie z użyciem protokołu w wersji 2.

`-4`
: Parametr wymusza połączenie z wykorzystaniem jedynie adresów IPv4.

`-6`
: Ten parametr z kolei wymusza połączenie z wykorzystaniem adresów IPv6.

`-A`
: Umożliwia połączenie ze specjalnie skonfigurowaną usługą identyfikacji użytkownika.

`-a`
: Opcja ta zamyka połączenie ze skonfigurowaną usługą identyfikacji użytkownika łączącego się z hostem.

`-b adresy`
: Parametr umożliwia zadeklarowanie adresu komputera, z którego mamy zamiar się połączyć. Opcja ta jest przydatna tylko wtedy, gdy komputer posiada więcej niż jeden adres IP.

`-C`
: Parametr włącza kompresję danych podczas połączenia. Rodzaj kompresji jest taki sam jak przy użyciu komendy `gzip`.

`-c szyfrowanie`
: Dzięki temu parametrowi wybieramy metodę szyfrowania przesyłanych danych.

`-D [adres:] port`
: Parametr ten umożliwia — oprócz wyboru adresu, z którym mamy zamiar się połączyć — wybór portu, na którym będziemy korzystać z serwera. Domyślnie jest to port `22`, jednak można go zmienić w konfiguracji systemu.

`-e znak`
: Definiuje znak wyjścia z sesji. Domyślny znak to tylda: „~”. Jeżeli po tym znaku postawimy kropkę „.”, wówczas połączenie zostanie zamknięte. Znak ten zostanie rozpoznany jedynie wówczas, gdy pojawi się jako pierwszy w linii podczas połączenia.

`-F plik_konfiguracyjny`
: Parametr ten pozwala na określenie specjalnego pliku przechowującego dane konfiguracyjne połączenia.

`-f`
: Umożliwia wprowadzanie danych w tle działania programu. Ma to zastosowanie np. podczas wprowadzania hasła dla danego serwera.

`-g`
: Parametr pozwala na korzystanie z portów lokalnych podczas połączenia.

`-I położenie`
: Umożliwia kontakt z zewnętrznym nośnikiem danych w celu pobrania klucza *RSA* podczas korzystania z sesji przez użytkownika.

`-i plik`
: Parametr ten umożliwia podanie lokalizacji pliku, w którym przechowywany jest klucz *RSA*.

`-K`
: Opcja umożliwia identyfikację użytkownika opartą na *GSSAPI*.

`-k`
: Opcja ta wyłącza identyfikację na podstawie *GSSAPI*.

`-L [adres:] port : host : port_nasłuchiwania`
: Pozwala na nasłuchiwanie na danym porcie i wykonywanie poleceń poprzez ten port. Ma to zastosowanie szczególnie wtedy, gdy dany port jest portem bezpiecznym.

`-l nazwa_użytkownika`
: Definiuje nazwę użytkownika, który łączy się z danym serwerem.

`-M`
: Nawiązuje połączenie z serwerem, ustawiając tryb tego połączenia na „master”.

`-m kod_autoryzacji`
: Dzięki temu parametrowi możemy zdefiniować kody autoryzacji MAC; jeżeli jest ich więcej niż jeden, należy oddzielić

je przecinkami. Opcja ta działa jedynie wtedy, gdy łączymy się z serwerem z użyciem protokołu w wersji 2.

`-N1`

Wstrzymuje realizację poleceń wydawanych z komputera połączonego z serwerem. Działa jedynie z protokołem w wersji 2.

`-n`

Przekierowuje informacje *stdin* z */dev/null*. Opcja ta może być użyta tylko wtedy, gdy `ssh` działa w tle.

`-O komenda`

Kontroluje aktywne połączenie „master" oraz jego procesy.

`-o opcja`

Parametr ten pozwala na ustawienie opcji podczas połączenia. Opcjami tymi są:

```
AddressFamily, BatchMode, BindAddress,
ChallengeResponseAuthentication, CheckHostIP, Cipher,
Ciphers, ClearAllForwardings, Compression,
CompressionLevel, ConnectionAttempts, ConnectTimeout,
ControlMaster, ControlPath, DynamicForward, EscapeChar,
ExitOnForwardFailure, ForwardAgent, ForwardX11,
ForwardX11Trusted, GatewayPorts, GlobalKnownHostsFile,
GSSAPIAuthentication, GSSAPIDelegateCredentials,
HashKnownHosts, Host, HostbasedAuthentication,
HostKeyAlgorithms, HostKeyAlias, HostName,
IdentityFile, IdentitiesOnly, KbdInteractiveDevices,
LocalCommand, LocalForward, LogLevel, MACs,
NoHostAuthenticationForLocalhost,
NumberOfPasswordPrompts, PasswordAuthentication,
PermitLocalCommand, Port, PreferredAuthentications,
Protocol, ProxyCommand, PubkeyAuthentication, RDomain,
RekeyLimit, RemoteForward, RhostsRSAAuthentication,
RSAAuthentication, SendEnv, ServerAliveInterval,
ServerAliveCountMax, SmartcardDevice,
StrictHostKeyChecking, TCPKeepAlive, Tunnel,
TunnelDevice, UsePrivilegedPort, User,
UserKnownHostsFile, VerifyHostKeyDNS, VisualHostKey,
XAuthLocation.
```

`-p port`
: Parametr definiuje port, za pomocą którego mamy zamiar połączyć się z serwerem.

`-q`
: Włącza tryb cichy. Wszystkie mało znaczące uwagi dotyczące błędów nie są wyświetlane.

`-R [adres:] port : host : docelowy_port`
: Połączenie zostanie przekazane na bezpieczny port na serwerze.

`-S gniazdo`
: Opcja pozwala na zdefiniowanie gniazda dla połączenia.

`-s`
: Parametr służy do wywołania podsystemu na docelowym serwerze.

`-T`
: Wyłącza alokację `pseudo-tty`.

`-t`
: Wymusza alokację typu `pseudo-tty`. Polecenie jest przydatne, kiedy wywołujemy na serwerze określoną aplikację.

`-V`
: Parametr powoduje wyświetlenie wersji polecenia.

`-v`
: Opcja powoduje włączenie wszystkich informacji o błędach.

`-w lokalny [:serwer]`
: Wymusza specjalny tryb tunelowania danych pomiędzy użytkownikiem i serwerem.

`-X`
: Umożliwia przekazywanie informacji ze środowiska *X11*.

`-x`
: Opcja ta wyłącza przekazywanie informacji z *X11*.

`-Y`
: Włącza zaufane przekazywanie informacji ze środowiska *X11*.

`-y`
: Wysyła logi informacji do logów systemowych za pomocą polecenia `syslog`.

Historia poleceń użytych w powłoce

Bardzo przydatne jest wywoływanie historii poleceń wykonywanych przez nas w powłoce. Służy do tego polecenie `history`.

W celu wypisania całej historii wystarczy użyć samego polecenia bez żadnych parametrów.

```
[lukasz@localhost ~]$ history
   1  ssh
   2  dir
   3  cd ~
   4  dir
```

`liczba`
: Jeżeli chcemy uzyskać określoną liczbę ostatnio wykonywanych poleceń, wpisujemy tę liczbę po poleceniu `history`.

```
[lukasz@localhost ~]$ history 2
   40  history
   41  history 2
```

`-c`
: Historię powłoki można usunąć, dodając specjalny parametr na końcu polecenia.

```
[lukasz@localhost ~]$ history -c
```

`!!`
: Ostatnie wprowadzone przez nas polecenie możemy wywołać przez zastosowanie dwóch wykrzykników po komendzie.

`!numer`
: Polecenie w historii możemy wywołać, dodając znak wykrzyknika, a następnie jego numer.,

`!-numer`
: Wywołuje polecenie, które zostało wpisane daną liczbę numerów wcześniej.

`!*`
: Ten argument powoduje wyświetlenie wszystkich parametrów poprzedniego polecenia.

Wypisywanie pierwszych wierszy pliku

Można w łatwy sposób, bez otwierania pliku, wypisać jego pierwsze linie. Ułatwia to znacznie wyszukiwanie potrzebnych informacji. Polecenie `head` wraz z podaniem nazwy pliku, bez żadnych parametrów, wypisze pierwsze 10 linii.

```
[lukasz@localhost ~]$ head plik.txt
linia 1
linia 2
linia 3
linia 4
linia 5
linia 6
linia 7
linia 8
linia 9
linia 10
```

`-liczba`
: Parametr powoduje wypisanie dowolnej zadeklarowanej w pozycji `liczba` linii plików. W przypadku zbyt długich linii ułatwia to nam czytanie zawartości pliku.

```
[lukasz@localhost ~]$ head -5 plik.txt
linia 1
linia 2
linia 3
linia 4
linia 5
```

`-c bajty`
: Wypisanie pierwszych bajtów pliku także jest możliwe. Wystarczy użyć parametru `-c`, po którym należy zdefiniować liczbę bajtów.

```
[lukasz@localhost ~]$ head -c 10 plik.txt
linia 1
```

`-q`
: Dzięki temu parametrowi możemy wypisać pierwsze linie pliku, ale bez nagłówka samego pliku.

```
[lukasz@localhost ~]$ head -q plik.txt
linia 1
linia 2
linia 3
linia 4
linia 5
linia 6
linia 7
linia 8
linia 9
linia 10
```

Wypisywanie ostatnich linii pliku

Można także wypisać ostatnie linie pliku dzięki poleceniu `tail` podanym z nazwą pliku. Ułatwia to odnalezienie poszukiwanych informacji. Domyślnie po wywołaniu tego polecenia zostanie wypisanych 10 linii.

`-liczba`
: Powoduje wypisanie określonej liczby ostatnich linii z pliku.

```
[lukasz@localhost ~]$ tail -5 plik.txt
linia 19
```

`+liczba`
: Parametr powoduje wypisanie wszystkich wierszy oprócz określonej liczby pierwszych linii.

```
[lukasz@localhost ~]$ tail +15 plik.txt
linia 15
linia 16
linia 17
linia 18
linia 19
```

`-c bajty`
: Powoduje wypisanie ostatnich bajtów pliku.

```
[lukasz@localhost ~]$ tail -c 15 plik.txt
nia 19
```

`-f`
: Dzięki temu parametrowi podany przez nas plik jest cały czas otwarty.

```
[lukasz@localhost ~]$ tail -f plik.txt
linia 11
[…]
linia 20
[kursor]
```

`-q`
: Powoduje wypisanie ostatnich linii z pliku bez jego nagłówków.

```
[lukasz@localhost ~]$ tail -q plik.txt
linia 11
[…]
linia 19
```

Uzyskiwanie informacji o trybie tworzenia nowych plików i katalogów

Wszystkie tworzone przez nas pliki i katalogi mają odpowiednio ustawione prawa, które określają, czy właściciel, grupa, do której należy dany użytkownik, oraz inni użytkownicy mają do nich dostęp. W celu uzyskania takich informacji należy wpisać polecenie `umask`.

```
[lukasz@localhost ~]$ umask
0022
```

Więcej informacji o trybie, w jakim polecenie wyświetla informacje, można znaleźć w opisie polecenia `chmod` w podręczniku *MAN*.

Wyświetlanie atrybutów plików i katalogów

Do wyświetlania atrybutów plików i katalogów służy polecenie `stat`, po którym należy podać nazwę pliku lub katalogu, którego właściwości chcemy zobaczyć.

```
[lukasz@localhost ~]$ stat plik.txt
  File: `plik.txt'
  Size: 0               Blocks: 0          IO Block: 4096
  ↪regular empty file
Device: 2bh/43d Inode: 6953848     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid:
↪(    0/    root)
Access: 2009-12-12 13:49:03.000000000 +0000
Modify: 2009-12-12 12:56:41.000000000 +0000
Change: 2009-12-12 12:56:41.000000000 +0000
```

`-l`

Parametr powoduje wyświetlenie dowiązania symbolicznego, do jakiego prowadzi dany plik.

`-f`

Wyświetla informacje na temat systemu plików, w którym znajduje się plik zdefiniowany w parametrze.

```
[lukasz@localhost ~]$ stat -f plik.txt
  File: "plik.txt"
    ID: 0        Namelen: 255     Type: ext2/ext3
Block size: 4096       Fundamental block size: 4096
Blocks: Total: 2621440    Free: 2411164    Available:
↪2411164
Inodes: Total: 5242880    Free: 5207838
```

`-t`

Wypisuje najważniejsze informacje w jednej linii.

```
[lukasz@localhost ~]$ stat -t plik.txt
plik.txt 0 0 81a4 0 0 2b 6953848 1 0 0 1260625743
↪1260622601 1260622601 4096
```

Dodatkowe prawa dostępu do plików

Poza standardowymi ustawieniami, za pomocą polecenia `chmod` można w Linuksie ustawiać także inne prawa specyficzne dla tego systemu. Służy do tego polecenie `chattr`. Pliki, dla których chcemy ustawić dodatkowe atrybuty, powinny znajdować się na partycjach *ext2* lub nowszych typach tego systemu plików.

W tym poleceniu przed prawami deklarujemy, czy chcemy je dodać (znakiem „+") czy odjąć (znakiem „-"), czy też ustawić bezwzględnie znakiem „=".

Dostępne atrybuty to:

- `a` — dołączanie danych,
- `A` — dostęp do pliku nie zmienia jego czasu ostatniego dostępu,
- `c` — plik zostanie skompresowany,
- `d` — informacja dla programu *dump*, aby pomijał ten plik przy tworzeniu kopii bezpieczeństwa,
- `i` — plik nie może być zmieniony lub usunięty (nie dotyczy konta *root*),
- `j` — umieszcza dane w dzienniku,
- `s` — jeżeli wybrany plik zostanie usunięty, wówczas miejsce na dysku, które on zajmował, zostanie zapisane zerami,
- `S` — wszystkie zmiany w pliku są natychmiast zapisywane na dysku,
- `u` — plik nie może zostać usunięty, nawet przez konto *root*,

Dodanie do pliku atrybutu, który nie zmienia czasu ostatniego dostępu:

```
[lukasz@localhost ~]$ chattr +A plik.txt
```

`-R`

Można także zmieniać prawa do wszystkich plików w katalogu. Wystarczy jedynie dodać opcję `-R`, wówczas wszystkie pliki, które znajdują się w katalogu podanym jako parametr, otrzymają takie prawa.

```
[lukasz@localhost ~]$ chattr -R +A katalog
```

Sprawdzanie dodatkowych uprawnień do plików

System Linux umożliwia także sprawdzenie dodatkowych atrybutów ustawionych dla pliku za pomocą polecenia `lsattr`.

```
[lukasz@localhost ~]$ lsattr plik.txt
-----a-A---------- plik.txt
```

`-R`

Opcja powoduje wypisanie wszystkich atrybutów plików, które znajdują się w katalogu (podanym jako parametr).

```
[lukasz@localhost ~]$ lsattr katalog
-------A---------- katalog/plik3.txt
-------A---------- katalog/plik2.txt
-------A---------- katalog/plik.txt
```

Wyszukiwanie danych w plikach

Jednym z najbardziej pomocnych narzędzi w systemie Linux jest polecenie `grep`, które pozwala na przeszukiwanie plików zawierających podany ciąg znaków.

Po poleceniu `grep` podajemy wyszukiwany przez nas ciąg znaków, a następnie plik lub grupę plików.

Aby za pomocą tego polecenia wyszukać w pliku *plik.txt* liczbę `12`, należy zapisać liczbę `12` po poleceniu, a następnie podać nazwę pliku.

```
[lukasz@localhost ~]$ grep 12 plik.txt
linia 12
```

Zostanie wyświetlona cała linia, która zawiera poszukiwany przez nas ciąg.

Polecenie to umożliwia stosowanie wyrażeń regularnych. Dzięki temu będziemy mogli lepiej wykorzystać jego właściwości.

Pojedynczy znak

Opcja umożliwia zadeklarowanie jednego znaku, który ma znajdować się w pliku. W wyniku zostaną wyświetlone wszystkie linie zawierające ten znak.

```
[lukasz@localhost ~]$ grep 1 plik.txt
linia 1
linia 10
linia 11
linia 12
linia 13
linia 14
linia 15
linia 16
linia 17
linia 18
linia 19
```

Dowolny pojedynczy znak z listy – `[…]`

Możemy także zadeklarować różne znaki — zostaną wówczas wyświetlone wszystkie linie, które je zawierają. W poleceniu użyliśmy liczb `1` i `6`. W wyniku znajdują się wszystkie linie, które spełniają powyższy warunek.

```
[lukasz@localhost ~]$ grep [16] plik.txt
linia 1
linia 6
linia 10
linia 11
linia 12
linia 13
linia 14
linia 15
linia 16
linia 17
linia 18
linia 19
```

Dowolny pojedynczy znak, który nie znajduje się na liście `[^...]`

Dzięki tej opcji możemy wybierać tylko te linie, które nie zawierają określonych znaków zadeklarowanych przez nas w poleceniu.

```
[lukasz@localhost ~]$ grep [^1] plik.txt
linia 2
linia 3
linia 4
linia 5
linia 6
linia 7
linia 8
linia 9
linia 2O
```

Wyszukiwanie na początku wiersza - `^`

Parametrem, który umożliwia wyszukiwanie jedynie na początku wiersza, jest znak `^`. Spowoduje on wypisanie wyników pasujących do podanej przez nas frazy i znajdujących się na samym początku wiersza.

```
[lukasz@localhost ~]$ grep ^specjalna plik2.txt
specjalna linia
```

Wyszukiwanie na końcu wiersza - `$`

Istnieje też parametr pozwalający na wyszukiwanie w pliku, który będzie sprawdzał zakończenie danej linii.

```
[lukasz@localhost ~]$ grep linia$ plik2.txt
specjalna linia
nowa linia
```

Można także sprawdzać, czy dane wyrażenie znajduje się na początku słowa, przez zastosowanie znaków `\<` oraz (na końcu danego słowa) znaków `\>`.

`[:alum:]`

Umożliwia wyszukanie dowolnego znaku alfanumerycznego.

```
[lukasz@localhost ~]$ grep [[:alnum:]] plik2.txt
linia 1
linia 2
linia 3
specjalna linia
nowa linia
```

`[:alpha:]`
: Wyszukuje dowolny znak alfabetu.

```
[lukasz@localhost ~]$ grep [[:alpha:]] plik2.txt
linia 1
linia 2
linia 3
specjalna linia
nowa linia
```

`[:cntrl:]`
: Parametr, dzięki któremu odnajdziemy dowolny znak kontrolny występujący w pliku.

```
[lukasz@localhost ~]$ grep [[:cntrl:]] plik2.txt
```

`[:digit:]`
: Wyszukuje dowolną cyfrę.

```
[lukasz@localhost ~]$ grep [[:digit:]] plik2.txt
linia 1
linia 2
linia 3
```

`[:graph:]`
: Umożliwia wyszukanie znaku graficznego.

```
[lukasz@localhost ~]$ grep [[:graph:]] plik2.txt
linia 1
linia 2
linia 3
specjalna linia
nowa linia
```

`[:lower:]`
: Wyszukuje dowolną małą literę występującą w pliku.

```
[lukasz@localhost ~]$ grep [[:lower:]] plik2.txt
linia 1
```

```
linia 2
linia 3
specjalna linia
nowa linia
```

`[:print:]`
: Szuka dowolnego znaku drukowalnego.

```
[lukasz@localhost ~]$ grep [[:print:]] plik2.txt
linia 1
Linia 2
Linia 3
specjalna linia
nowa linia
```

`[:punct:]`
: Wyszukuje dowolny znak interpunkcyjny.

```
[lukasz@localhost ~]$ grep [[:punct:]] plik2.txt
nowa linia.
```

`[:space:]`
: Wyszukuje dowolny znak biały (spacja, tabulator).

```
[lukasz@localhost ~]$ grep [[:space:]] plik2.txt
linia 1
Linia 2
Linia 3
specjalna linia
nowa linia.
```

`[:upper:]`
: Wyszukuje w pliku linie, w których występuje duża litera.

```
[lukasz@localhost ~]$ grep [[:upper:]] plik2.txt
Linia 2
Linia 3
```

`[:xdigit:]`
: Przeszukuje plik i wyświetla linie zawierające liczby heksadecymalne.

```
[lukasz@localhost ~]$ grep [[:xdigit:]] plik2.txt
```

Istnieją także różnice pomiędzy podstawowymi i rozszerzonymi wyrażeniami regularnymi.

Tabela 1. Podstawowe i rozszerzone wyrażenia regularne

Podstawowy	Rozszerzony	Znaczenie
`|`	`|`	Suma logiczna `Lub`.
`\+`	`+`	Jedno lub więcej powtórzeń wyrażenia regularnego.
`\?`	`?`	Zero lub jedno powtórzenie wyrażenia regularnego.
`\{n\}`	`{n}`	Dokładnie `n` powtórzeń wyrażenia regularnego.
`\{n,\}`	`{n}`	`n` lub większa liczba powtórzeń wyrażenia regularnego.
`\{n,m\}`	`{n,m}`	Od `n` do `m` powtórzeń wyrażeń regularnych.

Polecenie to zawiera także kilka pożytecznych opcji, dzięki którym można jeszcze bardziej dostosować wynik do naszych potrzeb.

`-v`

Powoduje wyszukanie tylko tych wierszy, które nie pasują do wyrażenia regularnego. Wyszukamy linie, w których nie występują duże litery:

```
[lukasz@localhost ~]$ grep [[:upper:]] -v plik2.txt
linia 1
specjalna linia
nowa linia.
```

`-l`

Powoduje wypisanie nazw plików, które zawierają podane wyrażenie regularne. Atrybut ten nie wypisze zawartości plików.

```
[lukasz@localhost ~]$ grep [[:alnum:]] -lr katalog
katalog/plik2.txt
```

`-L`

Wypisuje nazwy plików, które nie zawierają podanego wyrażenia regularnego. Atrybut ten nie spowoduje wypisania zawartości plików.

```
[lukasz@localhost ~]$ grep [[:alnum:]] -Lr katalog
katalog/plik3.txt
katalog/plik.txt
```

`-c`

Wyszukuje i wypisuje jedynie liczbę wierszy spełniających zgodność z podanym wyrażeniem regularnym.

```
[lukasz@localhost ~]$ grep [[:alnum:]] -c plik2.txt
5
```

`-n`

Przeszukuje plik pod względem występowania w nim wyrażenia regularnego i przed każdą linią wypisuje jej numer.

```
[lukasz@localhost ~]$ grep [[:alnum:]] -n plik2.txt
1:linia 1
2:Linia 2
3:Linia 3
4:specjalna linia
5:nowa linia.
```

`-b`

Wypisuje wynik przesunięcia w bajtach danego wiersza, pasujący do wyrażenia regularnego.

```
[lukasz@localhost ~]$ grep [[:alnum:]] -b plik2.txt
0:linia 1
8:Linia 2
16:Linia 3
24:specjalna linia
40:nowa linia.
```

`-i`

Powoduje, że w czasie sprawdzania pliku wielkość liter nie jest uwzględniana.

```
[lukasz@localhost ~]$ grep [[:upper:]] -i plik2.txt
linia 1
Linia 2
Linia 3
specjalna linia
nowa linia.
```

`-w`

Dzięki temu parametrowi będą brane pod uwagę jedynie całe słowa spełniające dane wyrażenie regularne.

```
[lukasz@localhost ~]$ grep [[:upper:]] -w plik2.txt
```

`-x`

W wyszukiwaniu w pliku brane są pod uwagę jedynie całe wiersze spełniające dane wyrażenie regularne. Jeżeli zastosujemy ten parametr razem z opcją `-w`, wówczas opcja `-x` opcja nie zostanie zastosowana.

```
[lukasz@localhost ~]$ grep [[:upper:]] -x plik2.txt
```

`-A liczba`

Podczas wyszukiwania dla każdego wiersza, który spełnia dane wyrażenie regularne, program wypisuje także zawartość linii następujących po tej, w której wyrażenie zostało spełnione.

```
[lukasz@localhost ~]$ grep [[:upper:]] -A 2 plik2.txt
Linia 2
Linia 3
specjalna linia
nowa linia.
```

`-B liczba`

Wyszukuje wiersze zgodne z wyrażeniem regularnym oraz wypisuje zawartość wierszy występujących przed linią, która pasuje do wyrażenia.

```
[lukasz@localhost ~]$ grep [[:upper:]] -B 2 plik2.txt
linia 1
Linia 2
Linia 3
```

`-C liczba`

Wyszukuje wiersze, które pasują do wyrażenia regularnego, oraz wyświetla wiersze znajdujące się w określonej odległości od pasującego wyrażenia. Wypisuje wiersze zarówno przed, jak i za wyszukiwanym wierszem.

```
[lukasz@localhost ~]$ grep [[:upper:]] -C 2 plik2.txt
linia 1
Linia 2
Linia 3
specjalna linia
nowa linia.
```

`-r`

Powoduje rekurencyjne wyszukiwanie plików znajdujących się w katalogu i podkatalogach, które zawierają dane wyrażenie regularne.

```
[lukasz@localhost ~]$ grep [[:alpha:]] -r katalog
katalog/plik2.txt:linia
```

Skorowidz

A

adduser, 96, 97
administracja systemem, 93
adres IP zdalnego komputera, 90
aktualizacja daty i czasu, 77
aktualna ścieżka, 70
alias, 52
aliasy, 51
apropos, 99
arch, 71
argumenty pobierane z wiersza powłoki, 117
atrybuty plików, 127

B

bash, 53
baza danych whatis, 99
break, 116

C

cal, 75
case, 114
cat, 39
cd, 34
chattr, 128, 129
chmod, 48, 49, 101
chown, 55
clear, 68
cmp, 63
continue, 117
cp, 43
czas, 77
czas działania systemu, 92
czas modyfikacji pliku, 42
czyszczenie terminala, 68

D

data, 77
data dostępu do pliku, 40
data modyfikacji pliku, 40
date, 77
demony usług, 94
    poziom działania, 95
df, 64
dir, 25
dodatkowe prawa dostępu, 128
dodawanie użytkownika, 96
dostępność hosta, 91
dowiązania symboliczne, 50
du, 65
dyski, 21
dystrybucje, 9

E

echo, 39, 103, 104
etc/init.d, 94
etc/rc.d, 95

F

FALSE, 106
fi, 113
file, 54
find, 56
finger, 73
for, 115
free, 74

G

Gnome, 14
grep, 129
groupadd, 99
groupdel, 99
grupa pliku, 55
grupy, 99

H

halt, 17
hasła, 52
head, 124
historia poleceń, 123
history, 123
home, 12, 20
host, 90
hostname, 87

I

IDE, 21
if, 112
ifconfig, 89
informacje o działających
  usługach, 95
informacje o pamięci
  systemowej, 74
informacje o sprzęcie, 71
informacje o typie pliku, 54
informacje o użytkowniku, 73
inittab, 93
instalacja systemu, 9
    dyskietki, 10
    opcje instalacji, 11
    partycje, 12
    płyty DVD, 10
interfejs sieciowy, 89

K

kalendarz, 75
katalogi, 23
    atrybuty, 127
    home, 20
    kopiowanie, 43
    mnt, 68
    prawa dostępu, 29, 48
    przenoszenie, 46
    tworzenie, 35
    usuwanie, 36
    wyświetlanie zawartości, 24
    zajętość miejsca, 65
    zmiana nazwy, 46
katalogi systemu, 19
KDE, 14
konto użytkownika, 97

kontrola wysyłania wiadomości, 82
kopiowanie katalogów i plików, 43

L

last, 83
less, 67, 68
Linux, 7, 8
listowanie zawartości katalogu, 24
ln, 51
login, 17
logout, 16
logowanie do systemu, 16
  tryb graficzny, 17
  tryb tekstowy, 16
ls, 24, 26
  parametry, 30
lsattr, 129

M

man, 22
Mandriva Linux, 9
MIME, 54
mkdir, 35
mnt, 68
montowanie systemu plików, 68
more, 67
mount, 69
mv, 46

N

nadawanie praw dostępu, 48
nazwy
  host systemowy, 87
  pliki, 23
  użytkownik, 86
  zdalny komputer, 90

O

odmontowanie systemu plików, 68
ostatnio zalogowani użytkownicy, 83

P

pamięć systemowa, 74
parametry interfejsu sieciowego, 89
partycje, 12, 21
  wolne miejsce, 64
passwd, 52
pętle, 115
ping, 91
pliki, 23
  atrybuty, 127
  data dostępu, 40
  data modyfikacji, 40
  dodatkowe prawa dostępu, 128
  grupa, 55
  ilość bajtów, słów i linii, 62
  informacje o typie, 54
  kopiowanie, 43
  porównywanie plików, 63
  prawa dostępu, 28, 48
  przenoszenie, 46
  tworzenie, 38
  ukryte, 24
  usuwanie, 38
  właściciel, 55
  wyszukiwanie, 56
  wyszukiwanie danych, 129
  wyświetlanie zawartości, 39
  zajętość miejsca, 65
  zmiana nazwy, 46
pomoc, 22
porównywanie plików, 63
powłoka, 53

poziom uruchomienia systemu, 93
prawa dostępu, 28, 128
  nadawanie, 48
  zapis numeryczny, 49
przechodzenie między katalogami, 34
przełączanie konta użytkownika, 70
przenoszenie katalogów i plików, 46
ps, 95
PuTTY, 118
pwd, 70

R

restart komputera, 17
rm, 38
rmdir, 36
root, 18, 20
rwall, 83

S

SCSI, 21
sh, 53
shutdown, 18
skrypty powłoki, 100
  argumenty wywołania, 117
  break, 116
  case, 114
  continue, 117
  for, 115
  if, 112
  pętle, 115
  test, 107
  until, 115
  uruchamianie, 101
  wartości logiczne, 106
  while, 115
  wypisywanie tekstu na ekranie, 103
  zmienne, 102
sprawdzanie
  dodatkowe uprawnienia do plików, 129
  dostępność hosta, 91
  własna nazwa użytkownika, 86
sprzęt, 71
ssh, 118
SSH, 118
stat, 127
strony MAN, 22
su, 71
swap, 12
system plików, 21

Ś

środowisko pracy, 14

T

tail, 125
Terminal, 15
test, 107
then, 113
touch, 38, 40
TRUE, 106
tryb tworzenia nowych plików, 126
tworzenie
  aliasy, 51
  dowiązania symboliczne, 50
  katalogi, 35
  pliki, 38
typ pliku, 54

U

umask, 126
umount, 70
uname, 71, 72
until, 115
uptime, 92
uruchamianie usługi, 94
userdel, 98
users, 86
usługi, 94
  informacje, 95
usuwanie
  katalogi, 36
  pliki, 38
  użytkownicy, 98
użytkownicy, 18
  dodawanie, 96
  informacje, 73
  katalog, 20
  root, 20
  usuwanie, 98

V

vdir, 25

W

wall, 83
wartości logiczne, 106
wc, 62
whatis, 99
while, 115
who, 86
whoami, 86
whois, 91
właściciel pliku, 55
wolne miejsce na partycjach, 64
write, 82
wylogowanie, 16
wyłączanie komputera, 17
wyrażenia regularne, 134
wysyłanie
  komunikaty do wszystkich sieci z pliku tekstowego, 83
  wiadomości, 82
  wiadomości z pliku tekstowego, 83
wyszukiwanie
  dane w plikach, 129
  pliki, 56
wyświetlanie
  ostatnie linie pliku, 125
  pierwsze wiersze pliku, 124
  tekst na ekranie, 103
  zawartość katalogu, 24
  zawartość pliku, 39

X

XFree, 14

Z

zalogowani użytkownicy, 74, 86
zarządzanie zasobami, 23
zasoby, 23
zatrzymywanie usługi, 94
zmiana
  grupa pliku, 55
  hasło, 52
  nazwa pliku, 46
  powłoka, 53
  właściciel pliku, 55
zmienne, 102
znak zachęty, 24