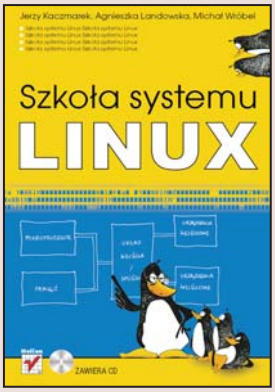
Szkoła systemu Linux



Autorzy: Jerzy Kaczmarek, Agnieszka Landowska, Michał Wróbel
ISBN: 83-246-0767-6
Format: B5, stron: 320

Coraz większą popularność w świecie oprogramowania zdobywają produkty typu open source z systemem operacyjnym Linux na czele. Dlaczego i Ty powinieneś dołączyć do wciąż rosnącej społeczności użytkowników systemów z tej rodziny? Ponieważ zapewniają bezpieczne, stabilne oraz darmowe środowisko pracy. Dzięki wielu dystrybucjom systemu Linux zarówno początkujący, jak i zaawansowani użytkownicy mogą wybrać produkt dostosowany do swych potrzeb.

Książka „Szkoła systemu Linux" szybko wprowadzi Cię w świat oprogramowania typu open source. Poznasz liczne programy, które umożliwiają między innymi zarządzanie danymi, pisanie i drukowanie dokumentów czy korzystanie z sieci. Nauczysz się używać środowiska graficznego oraz wiersza poleceń. Dowiesz się, jak działają systemy operacyjne, w tym Linux. Na płycie dołączonej do książki znajduje się jedna z dystrybucji systemu Linux, cdlinux.pl, którą możesz uruchomić bezpośrednio z napędu CD-ROM. Dzięki temu będziesz mógł szybko i bezpiecznie wykonać opisane ćwiczenia oraz utrwalić nabytą wiedzę.

- Uruchamianie systemu Linux
- Korzystanie z programów biurowych
- Praca w środowisku graficznym
- Korzystanie z wiersza poleceń
- Aplikacje związane z internetem
- System plików i zarządzanie danymi
- Programy multimedialne
- Systemy zarządzania treścią (Joomla! i Mambo)
- Zarządzanie dyskami
- Instalowanie systemu Linux

Poznaj system Linux dzięki 40 prostym lekcjom

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 7.

Zarządzanie systemem z wiersza poleceń

Najważniejszym zadaniem każdego systemu operacyjnego jest uruchamianie programów użytkowych. Na komputerze można uruchamiać różnego rodzaju programy. Najczęściej są to aplikacje, takie jak edytory, programy do odtwarzania plików multimedialnych, przeglądarki internetowe czy gry komputerowe. Inne specjalizowane programy mogą umożliwiać zarządzanie bazami danych czy projektowanie domów. Programy tego typu nie są bezpośrednio związane z pracą systemu operacyjnego — są jedynie przez niego uruchamiane, kontrolowane i zarządzane.

Istnieje również liczny zbiór programów przeznaczonych do kontroli działania systemu operacyjnego oraz do sterowania i zarządzania nim. Programy te służą do kontaktów użytkownika z systemem, do zarządzania plikami, procesami, pamięcią operacyjną, dyskami i do kontroli funkcji systemu. Stanowią one integralną część systemu operacyjnego i dlatego znajdują się w każdej dystrybucji systemu Linux. Wykonywalne programy do zarządzania systemem operacyjnym są w systemie Linux uruchamiane z wiersza poleceń i dlatego często nazywa się je również poleceniami powłoki.

W wierszu poleceń można wydać wiele rozkazów do zarządzania procesami, plikami i innymi mechanizmami w systemie operacyjnym. Programy do zarządzania systemem można przydzielić, w zależności od ich przeznaczenia, do następujących kategorii:

- zarządzanie użytkownikami: `passwd`, `adduser`, `su`,
- zarządzanie procesami: `ps`, `kill`, `bg`, `fg`,
- przetwarzanie tekstu: `grep`, `egrep`, `wc`, `cut`, `diff`, `tr`, `sed`,
- definiowanie środowiska pracy: `alias`, `unalias`, `umask`,
- operacje na plikach: `chmod`, `touch`, `find`,
- zarządzanie systemem plików: `cd`, `cp`, `ls`, `mv`, `rm`, `mkdir`.

Nazwy podstawowych programów do zarządzania i komunikacji z systemem operacyjnym są zawsze takie same we wszystkich systemach operacyjnych typu Linux lub Unix. Ich możliwości mogą być jednak różne w zależności od ich wersji.

Programy do zarządzania systemem operacyjnym znajdują się między innymi w katalogu */sbin*, jednak aby je uruchomić, wystarczy w wierszu poleceń wpisać ich nazwę, bez pełnej ścieżki dostępu. Jest to możliwe dzięki ustawieniom środowiska pracy użytkownika, wśród których znajduje się zmienna wyznaczająca katalogi poszukiwania programów. Domyślnie w tej zmiennej znajduje się również katalog */bin*. Dzięki temu z punktu widzenia użytkownika programy do zarządzania systemem operacyjnym mogą być traktowane jako polecenia powłoki, ponieważ są przez nią poprawnie interpretowane i wykonywane.

Dzięki programom do zarządzania systemem operacyjnym użytkownik może kontrolować system operacyjny i podejmować decyzje o stanie procesów, priorytetach ich uruchamiania. Ma także możliwość rozwiązywania problemów, jakie mogą się pojawić w trakcie pracy systemu operacyjnego.

Lekcja 30. Polecenia do zarządzania procesami

Mechanizmy tworzenia procesów i zarządzania nimi należą do złożonych i trudnych zagadnień w systemach operacyjnych. Ponieważ czasami dochodzi do zawieszenia procesów w wyniku niepoprawnych działań użytkownika lub błędów w programach, dlatego ważne jest zrozumienie, jak procesy powstają i działają.

Zarządzanie procesami w systemach operacyjnych bazuje na znanych z codzienności metodach organizacji i zarządzania działaniem. Człowiek organizuje swoją pracę intuicyjnie, rozróżniając rzeczy ważne i pilne oraz te mniej istotne. Jednak w przypadku zarządzania dokonywanego przez system komputerowy brakuje intuicji i inteligencji człowieka, dlatego programy do zarządzania procesami są bardzo złożone.

Jedną z analogii, która dobrze ilustruje zagadnienie zarządzania procesami, może być czynność przygotowania posiłku. Żeby zrobić obiad, trzeba wykonać wiele czynności, takich jak przygotowanie produktów, ich mycie, krojenie, gotowanie czy zmywanie naczyń. Kolejność tych czynności nie zawsze może być dowolna. Najpierw trzeba marchewkę obrać, a następnie dodać do zupy. Podobnie w systemie operacyjnym, żeby wykonać pewne działanie, np. przetwarzanie danych, konieczne jest wykonanie wielu czynności: otwarcie pliku, jego odczytanie, przekształcenie danych i zapis do pliku. Za te czynności mogą odpowiadać różne, niezależne procesy i podobnie jak w przypadku gotowania pewne czynności muszą zostać wykonane w określonej kolejności. W systemach operacyjnych za kolejność wykonywania procesów odpowiadają wyspecjalizowane programy kolejkowania i szeregowania zadań.

Jeżeli kucharz jest jeden, to musi dzielić swój czas pomiędzy różne czynności. W czasie operacji krojenia, gdy występuje zdarzenie wymagające interwencji kucharza, np. kipi zupa, trzeba przerwać krojenie i zamieszać w garnku. Jednak częste przełączanie się między czynnościami powoduje, że czas ich realizacji znacznie się wydłuża. Gdy czas przełączania się między czynnościami przekracza czas poświęcany na realizację zadań, mamy do czynienia ze zjawiskiem nazywanym w systemach operacyjnych szamotaniem. Czasem lepiej zakończyć jeden proces, aby rozpocząć drugi. W systemach komputerowych odpowiednikiem kucharza jest procesor. To on odpowiada za realizację procesów, pomiędzy którymi jest przełączany. Jednym z mechanizmów szeregowania zadań jest przydzielanie kwantu (przedziału) czasu na realizację danego procesu. Innym mechanizmem szeregowania zadań jest nadawanie procesom priorytetów, oznaczających ich ważność. Podobnie jak w przypadku gotowania — kipiąca zupa przerywa inne czynności, np. krojenie. W systemach operacyjnych np. czynności zarządzania procesami mają wyższy priorytet niż uruchomienie gry dla użytkownika. Złożone algorytmy kolejkowania i zarządzania zadaniami zmierzają do eliminacji zjawiska szamotania.

Warto sobie uświadomić, że przy wszystkich procesach przygotowania potraw korzystamy z tych samych zasobów: kuchni, mebli, oświetlenia, noży, śmietnika itp. Zasoby te są współdzielone, tak jak kucharz, i również są przełączane pomiędzy procesami. W systemie komputerowym współdzielony jest procesor, ale także pamięć operacyjna, zasoby dyskowe i inne urządzenia. Należy podkreślić, że w systemie operacyjnym w każdej chwili wykonywany jest tylko jeden proces, który na ten czas otrzymuje wyłączny dostęp do wszystkich potrzebnych mu zasobów. Po zakończeniu kwantu czasu procesor i inne zasoby są przekazywane do dyspozycji kolejnego procesu.

W przypadku gdy komputer ma jeden procesor, czynności zarządzania procesami przebiegają inaczej niż w systemach wieloprocesorowych. Podobnie jak w przysłowiu: „gdzie kucharek sześć, tam nie ma co jeść", jeżeli procesorów jest wiele, to mamy do czynienia z problemami współbieżności, współdzielenia zasobów i rozproszenia przetwarzania. Obsługa komputerów z wieloma procesorami przez system operacyjny jest bardziej skomplikowana, jednak z punktu widzenia użytkownika takie maszyny są bardziej wydajne i pozwalają na szybszą realizację złożonych i pracochłonnych zadań.

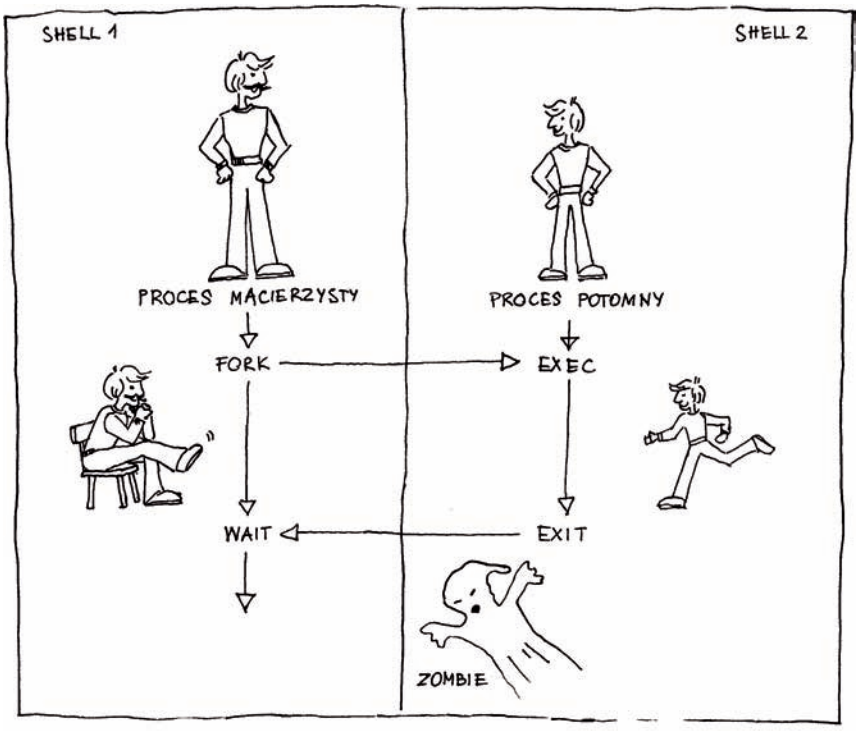
Kończąc te porównania, warto zdawać sobie sprawę, że podstawy algorytmów do zarządzania procesami w systemach operacyjnych mają swoje źródła w zarządzaniu czynnościami życia codziennego.

W systemie operacyjnym może pracować jednocześnie wielu użytkowników, którzy uruchamiają wiele różnych programów. Każdy z programów stanowi jeden lub więcej procesów. Proces może działać w **trybie użytkownika** (ang. *user mode*) lub **trybie jądra** (ang. *kernel mode*), z których ten ostatni ma wyższy poziom uprzywilejowania w korzystaniu z czasu procesora oraz dostępu do pamięci. Każdy nowy proces ma przypisany unikalny numer identyfikacyjny **PID** (ang. *Process IDentificator*). Numery PID są nadawane w kolejności tworzenia procesów. Informacje o procesie zapisane są w specjalnej strukturze nazywanej u-obszarem. Jest to zbiór danych zawierający takie informacje, jak numer identyfikacyjny **UID** (ang. *User ID*) osoby, która proces utworzyła, dane o terminalu użytkownika, bieżącym katalogu, plikach dostępnych dla procesu i wiele innych.

Proces może znajdować się w następujących stanach: nowy, gotowy do uruchomienia, aktywny (gdy przydzielono mu procesor) oraz zatrzymany, gdy czeka na wystąpienie jakiegoś zdarzenia, na przykład na zakończenie działania innego procesu. Dane o wszystkich uruchomionych w systemie procesach są zapisywane w specjalnej tablicy procesów. Znajdują się w niej takie informacje, jak identyfikator właściciela, rozmiar pamięci niezbędnej dla procesu oraz jego priorytet i stan. Na podstawie tych danych system operacyjny podejmuje decyzję, który proces zostanie uruchomiony jako następny. Jeżeli proces nie zakończył się w ustalonym kwancie czasu, to zostanie przerwany, zatrzymany i umieszczony w kolejce procesów oczekujących na wykonanie.

Mechanizm powstawania nowego procesu przebiega zawsze według pewnej procedury. Nowy proces może powstać jedynie poprzez powielenie innego, już istniejącego procesu. Powielany proces nazywany jest macierzystym (a potocznie rodzicem), natomiast nowo powstały proces — potomnym (potocznie dzieckiem).

Na rysunku 30.1 przedstawiono mechanizm powielenia i tworzenia procesów.



Rysunek 30.1. *Powstawanie nowego procesu uruchomienia programu*

Proces macierzysty wykonuje operację **powielenia** (ang. *fork*) i powstaje proces potomny. Nowy proces otrzymuje własny identyfikator PID, a także zapamiętuje identyfikator rodzica **PPID** (ang. *Parent PID*). Proces macierzysty zostaje zawieszony i czeka (ang. *wait*) na zakończenie procesu potomnego. W procesie potomnym dochodzi do wykonania (ang. *exec*) zawartego w nim kodu programu. Po zakończeniu (ang. *exit*) proces

potomny przekazuje do rodzica informację o prawidłowości wykonania, tak zwany kod wyjścia. Przy poprawnym wykonaniu procesu potomnego kod wyjścia jest liczbą zero, a gdy proces zakończy się niepoprawnie, jest to liczba niezerowa. Kody wyjścia można wykorzystywać do sterowania procesami i wysyłania komunikatów do użytkownika o nieprawidłowościach w wykonywanych programach. Końcowym stanem procesu potomnego jest stan zwany **zombie**, który oznacza, że proces już zakończył działanie, ale jeszcze czeka, aby proces macierzysty odebrał stworzone przez niego dane i kod wyjścia.

Przodkiem wszystkich procesów w systemie jest proces o nazwie **init** i identyfikatorze PID równym *1*, który jest tworzony w momencie startu systemu operacyjnego. Następnie uruchamiane są procesy obsługi sprzętu, a potem procesy logowania i identyfikacji użytkownika. Procesy te umożliwiają użytkownikowi zalogowanie się do systemu operacyjnego. Po zakończeniu tej operacji dla użytkownika jest uruchamiany proces powłoki. Proces pierwszej powłoki jest niezwykle ważny dla użytkownika, ponieważ każdy uruchamiany przez niego program może zaistnieć w systemie jedynie w wyniku jej powielenia. Powłoka staje się procesem macierzystym dla innych procesów użytkownika.

Tak jak dziecko dziedziczy pewne cechy rodzica, tak proces potomny dziedziczy cechy procesu macierzystego. Powielona powłoka przenosi do nowego procesu dane o użytkowniku, jego terminalu, prawa dostępu do otwartych plików, informacje o środowisku pracy, zdefiniowane aliasy i inne. Uruchamianie programu poprzez powłokę przypomina nieco inżynierię genetyczną, gdy do komórki macierzystej wprowadzamy dane zmieniające nieznacznie wygląd organizmu. Nowy organizm będzie miał wiele cech rodzica, ale też własne nowe cechy. Powielony proces powłoki staje się jakby komórką macierzystą, do której wprowadzamy dodatkowe wiersze kodu programu, który chcemy uruchomić. Tą czynność realizują funkcje `fork` i `exec`, czyli powiel i wykonaj.

Zarządzanie procesami przez użytkownika polega nie tylko na ich tworzeniu, ale również na ich zatrzymywaniu, uruchamianiu w określonym czasie i z określonym priorytetem czy usuwaniu. Wśród poleceń i mechanizmów do zarządzania procesami można wyróżnić:

- `ps` — wyświetlanie listy uruchomionych procesów,
- `kill` — usuwanie procesów,
- `jobs` — wyświetlanie listy procesów wykonywanych w tle,
- `&` — uruchamianie procesu w tle,
- `|` — przekazywanie danych z jednego do drugiego procesu,
- `nice` — uruchamianie procesu z niskim priorytetem,
- `nohup` — uruchamianie procesu, który będzie działał także po zakończeniu sesji przez użytkownika.

Zarządzanie procesami

Użytkownik może kontrolować procesy, które zostały przez niego uruchomione. Staje się to konieczne w sytuacji, gdy dochodzi do zawieszenia procesu lub gdy użytkownik nie wie, jak program zakończyć. Zatrzymanie jednego z procesów może być wynikiem błędu programu lub pomyłki użytkownika, jednak nie oznacza to, że system operacyjny przestał funkcjonować poprawnie. System operacyjny Linux zawiesza się niezwykle rzadko. Proces może być zawieszony, czyli czekać na zakończenie działania innego procesu lub na wystąpienie jakiegoś zdarzenia. Jednak w tym samym czasie pozostali użytkownicy systemu mogą bez przeszkód realizować własne zadania. W sytuacji zawieszenia procesu można za pomocą mechanizmów systemu operacyjnego wyeliminować go i powrócić do normalnej pracy.

W celu poznania mechanizmów zarządzania procesami wykonaj w trybie tekstowym następujące kroki:

1. Zaloguj się do systemu i wyświetl uruchomione procesy, wpisując w wierszu poleceń:

   ```
   # ps
   ```

   Polecenie `ps` wyświetla informacje o uruchomionych procesach. Wywołane bez parametru wyświetla tylko procesy uruchomione na bieżącym terminalu użytkownika. Zauważ, że pokazane są trzy procesy: proces logowania, proces bieżącej powłoki oraz właśnie uruchomiony proces realizujący wydane polecenie `ps`. Dwa pierwsze procesy istnieją dla każdego użytkownika na każdym jego terminalu i umożliwiają wydawanie poleceń. Dla każdego procesu, oprócz jego nazwy, podawany jest także numer PID oraz nazwa terminalu, na którym został uruchomiony.

2. Uruchom edytor `vim`, wpisując w wierszu poleceń:

   ```
   # vim plik1
   ```

   Uruchomienie nowego programu powoduje utworzenie przez system operacyjny nowego procesu użytkownika. Edytor `vim` ma duże możliwości i jest często używany przez administratorów systemów operacyjnych do pracy w trybie tekstowym. W tym trybie edytor zajmuje cały ekran i użytkownik nie ma możliwości wydawania poleceń do systemu, ponieważ nie ma dostępu do wiersza poleceń. Aby powrócić do wiersza poleceń, należy zakończyć proces edycji poprzez naciśnięcie pewnej kombinacji klawiszy. Ta kombinacja klawiszy nie jest intuicyjna i jeżeli użytkownik jej nie zna, bardzo trudno jest ją zgadnąć. Dodatkowo edytor `vim` w trakcie edycji nie podpowiada sposobu jej zakończenia. Istnieje nawet dowcip, mówiący o możliwości wygenerowania dowolnie długiego losowego ciągu znaków przez użytkownika, który próbuje wyjść z programu `vim`. Załóżmy, że nie jest znany sposób zakończenia pracy z edytorem `vim`.

3. Spróbuj zakończyć działanie programu `vim`, naciskając kombinację klawiszy *Ctrl+C*.

   Kombinacja ta umożliwia zazwyczaj przerwanie bieżącego procesu, lecz w przypadku edytora `vim` jest ona nieskuteczna. W takiej sytuacji jedną z dróg rozwiązania problemu jest usunięcie zawieszonego procesu.

4. Przejdź do innego terminalu za pomocą kombinacji klawiszy *Alt+F2*. Zaloguj się ponownie, używając tego samego konta użytkownika. Zauważ, że masz do czynienia z sytuacją, w której system operacyjny działa poprawnie. Brak możliwości wydawania poleceń na poprzednim terminalu wynika z jego zablokowania przez jeden z procesów.

5. Wyświetl informacje o zalogowanych w systemie użytkownikach, wpisując w wierszu poleceń:

   ```
   # finger
   ```

   Polecenie `finger` wydane bez parametrów umożliwia odczytanie informacji o wszystkich terminalach, na których pracują zalogowani użytkownicy. Z wyświetlonych informacji powinno wynikać, że jesteś użytkownikiem dwóch terminali — jednego, na którym właśnie pracujesz, oraz drugiego, w którym został uruchomiony edytor `vim`. Zauważ, że terminale w trybie tekstowym są oznaczane nazwą *tty* z numerem nadawanym w kolejności ich uruchamiania. W trybie graficznym oznaczenia terminali to *pts/0*, *pts/1* i następne.

6. Wyświetl informacje o uruchomionych dotychczas procesach, wpisując w wierszu poleceń:

   ```
   # ps -t tty1
   ```

   Polecenie `ps` z opcją `-t` wyświetla procesy uruchomione na terminalu, którego nazwa została podana jako parametr. Na jednym z używanych terminali wśród uruchomionych procesów znajduje się proces edytora `vim`. Odczytaj jego numer identyfikacyjny PID wyświetlony w pierwszej kolumnie, np. *4269*. Numer PID zawieszonego procesu jest niezbędny przy jego usuwaniu.

7. Zamknij program edytora `vim` poprzez likwidację jego procesu. W tym celu wpisz w wierszu poleceń:

   ```
   # kill -9 4269
   ```

   Polecenie `kill` (dosłownie: zabij) zamyka proces o numerze PID podanym jako parametr wywołania. Polecenie to wywołane tylko z numerem procesu bez opcji `-9` umożliwia procesowi samodzielne zakończenie działania, dzięki czemu możliwe jest np. zapisanie i zamknięcie plików tymczasowych. Wywołanie polecenia `kill` z opcją `-9` oznacza, że system operacyjny usunie proces niezależnie od jego stanu i priorytetu.

8. Przejdź do terminalu, na którym był uruchomiony edytor, i sprawdź, czy proces edytora został zakończony. Sprawdź listę procesów na terminalu za pomocą polecenia `ps`. Zauważ, że procesu edytora nie ma już na tej liście. Warto zapamiętać, że pracę w edytorze `vim` kończy wpisanie ciągu znaków `:q!` i naciśnięcie klawisza *Enter*.

Zarządzanie wieloma procesami

Użytkownik ma możliwość pracy z wieloma procesami jednocześnie. Niekiedy zachodzi konieczność, żeby w trakcie pracy z jednym programem uruchomić inny czy przejść do wiersza poleceń. Inne programy mogą być uruchamiane w tle, kiedy nie

zachodzi konieczność kontroli ich wykonania albo interakcji z użytkownikiem. W przypadku pracy z wieloma procesami będą one wykonywane w tle, co oznacza, że mogą one działać jednocześnie, umożliwiając dalszą pracę w wierszu poleceń. Jednak istnieje możliwość przełączenia się na dany proces w celu jego kontroli czy kontynuacji interakcji. Przełączanie się między procesami w trybie tekstowym jest podobne do pracy z wieloma oknami w trybie graficznym. Tylko jedno okno może być w danym momencie aktywne, a użytkownik ma możliwość przełączania się między oknami.

Aby pracować z wieloma procesami, wykonaj w trybie tekstowym następujące polecenia:

1. `# sleep 300`

   Uruchom program `sleep`, który tworzy proces czekający przez podaną jako parametr liczbę sekund. Dopiero po upływie zadanego czasu proces kończy swoje działanie i umożliwia dalszą pracę w wierszu poleceń. Naciśnij klawisze *Ctrl+Z*. Ta kombinacja klawiszy pozwala na zatrzymanie bieżącego procesu i przejście do wiersza poleceń.

2. `# fg`

   Polecenie `fg` (ang. *foreground*) umożliwia przywrócenie wykonywania procesu, który został wcześniej zatrzymany. Polecenie to wywołane bez parametru przywróci proces, który ostatnio został umieszczony w tle, czyli proces programu `sleep`. Ponownie zatrzymaj ten proces za pomocą kombinacji klawiszy *Ctrl+Z*.

3. `# vim plik1&`

   Uruchom edytor `vim` jako proces działający w tle. Znak *&* jest poleceniem dla powłoki, żeby poprzedzające go polecenie uruchomić w tle.

4. `# man more&`

   Uruchom jeszcze jeden proces w tle, np. wyświetlenie pomocy dla polecenia `more`.

5. `# jobs`

   Polecenie `jobs` umożliwia wyświetlenie procesów uruchomionych w tle. Przykładowy rezultat jego wykonania może wyglądać następująco:

   ```
   [1]       Stopped          sleep 300
   [2]  -    Stopped          vim plik1
   [3]  +    Stopped          man more
   ```

   Pierwsza kolumna zawiera numer procesu, którego można użyć przy przywracaniu podglądu działania procesu. Druga kolumna zawiera znaki + i – odpowiednio przy procesie na pierwszym i drugim planie. Na pierwszym planie jest proces ostatnio przeniesiony w tło. Polecenie `fg` wywołane bez parametrów uruchomi podgląd wykonania właśnie tego procesu. Trzecia kolumna zawiera opis stanu procesu. Wyświetlone procesy są w stanie zatrzymania (ang. *stopped*). W tym polu może się także pojawić wartość *Running* (działający) dla procesów, które nie wymagają interakcji z użytkownikiem. Ostatnia kolumna pokazuje nazwę programu i parametry jego wywołania.

6. `# bg %1`

Polecenie `bg` (ang. *background*) uruchamia w tle zatrzymany wcześniej proces, którego numer podano jako parametr. Wydane z parametrem `%1` oznacza, że wykonywanie procesu `sleep` zostało wznowione. Zauważ, że ruchomienie procesu w tle oznacza, że wykonuje on swoje działania, a użytkownik może w tym samym czasie wydawać kolejne polecenia w programie powłoki. Uruchom ponownie polecenie `jobs` i zauważ, że stan procesu `sleep` zmienił swoją wartość na *Running*. Po zakończeniu działania proces `sleep` wyświetli na ekranie komunikat o zakończeniu pracy.

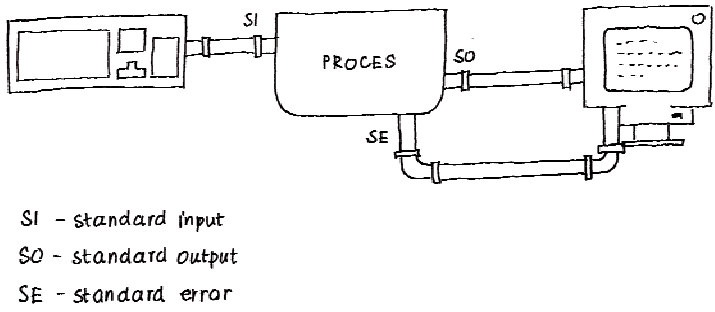
7. `# fg %3`

Polecenie `fg` przywraca działanie procesu. Wywołane bez parametru uruchomi proces znajdujący się w tle na pierwszym planie (oznaczenie +). Wywołane z numerem procesu przywraca podgląd jego działania, w podanym przykładzie wyświetli rezultat działania polecenia `man more`. Zakończ procesy `man` i `vim`, przełączając się na nie kolejno i zamykając programy. Procesy możesz też usunąć, korzystając z poznanego polecenia `kill`. Poczekaj na samodzielne zakończenie pracy przez proces `sleep`.

Standardowe strumienie procesów

Działanie każdego programu polega w pewnym sensie na przetwarzaniu danych, niezależnie od tego, czy te dane pochodzą z pliku, czy z klawiatury użytkownika. Do operacji przetwarzania konieczne są dane wejściowe oraz sposób prezentacji rezultatów. Przekazywanie danych przypomina przepływ strumienia wody i dlatego w informatyce mówi się o strumieniach danych. Każdy proces od chwili uruchomienia posiada trzy strumienie danych — wejściowy, wyjściowy i strumień błędów.

Ideę procesu i standardowych strumieni wejściowych i wyjściowych pokazuje rysunek 30.2.



Rysunek 30.2. *Proces i standardowe strumienie*

Każdy proces posiada **standardowe wejście SI** (ang. *standard input*), którym zwykle jest klawiatura. Innymi strumieniami wejściowymi mogą być otwarte dla procesu pliki. **Standardowym wyjściem SO** (ang. *standard output*) jest zwykle monitor, ale także może to być plik. Każdy proces posiada również **standardowy strumień błędów SE** (ang. *standard error*), za pomocą którego przekazywane są informacje o błędach

pojawiających się w trakcie trwania procesu. Komunikaty o błędach mogą pojawiać się na monitorze i zaciemniać dane wyjściowe programu. Metodą eliminacji tego problemu jest przekazywanie komunikatów o błędach do specjalnego pliku. Wszystkie strumienie procesu mają nadane tzw. deskryptory, które ułatwiają ich rozpoznawanie przez system operacyjny. Deskryptory są to liczby całkowite, i tak strumień wejściowy to *0*, wyjściowy to *1*, a strumień błędów to *2*.

W systemie operacyjnym występuje mechanizm przekierowania strumieni danych wejściowych i wyjściowych. Pozwala on na decydowanie, skąd pochodzą dane przekazywane do procesu i gdzie mają być umieszczone po jego zakończeniu. Zmiana kierunku strumienia wejściowego realizowana jest przez < (znak mniejszości), strumienia wyjściowego realizowana jest przez > (znak większości), a dopisanie danych do pliku realizuje znak >> (dwa znaki większości).

W celu wykorzystania mechanizmu przekierowania wpisz w wierszu poleceń:

1. `# cat > plik1`

   Polecenie `cat` służy do przeglądania i łączenia plików, ale może także służyć jako pseudoedytor.

   Ze względu na to, że nie podano w poleceniu strumienia wejściowego, standardowym wejściem jest klawiatura. Oznacza to, że znaki wpisywane przez użytkownika na klawiaturze zostaną przekazane do polecenia `cat`. W kolejnych wierszach wpisz po kilka znaków, niektóre z wierszy zacznij od litery `s`. W trakcie pisania wiersze tekstu będą również widoczne na monitorze.

   Znak > oznacza przekierowanie strumienia wyjściowego do pliku o podanej nazwie. Jeżeli plik wcześniej nie istniał, zostanie on utworzony. Wpisane przez użytkownika wiersze tekstu zostaną zapisane w tym pliku. Działanie polecenia `cat` może być zakończone klawiszami *Ctrl+D*. Podejrzyj zawartość utworzonego pliku za pomocą polecenia `more`.

2. `# cat <plik1 >plik2`

   W podanym poleceniu `cat` dzięki mechanizmowi przekierowania strumieniem wejściowym jest plik *plik1*, a strumieniem danych wyjściowych — *plik2*. Oznacza to, że zawartość jednego z plików zostanie bez modyfikacji przepisana do drugiego pliku. Sprawdź za pomocą polecenia `more` rezultat wykonania operacji.

3. `# cat >>plik2`

   W podanym poleceniu standardowym wejściem dla procesu `cat` ponownie jest klawiatura, natomiast standardowym wyjściem — *plik2*. Użycie podwójnego znaku większości >> oznacza, że wpisywane z klawiatury dane nie zastąpią dotychczasowych danych w pliku, ale zostaną dopisane na jego końcu. Wpisz kilka wierszy tekstu, niektóre zaczynając od litery `s`, zakończ edycję i sprawdź zawartość pliku za pomocą polecenia `more`.

4. `# cat plik4`

   Wywołaj polecenie `cat`, podając jako parametr nieistniejący plik. Zauważ pojawiający się komunikat o błędzie. Komunikaty o błędach mogą być

wyświetlane pomiędzy danymi wyjściowymi procesu, co zmniejsza czytelność wyników. Dlatego niekiedy wyświetlanie komunikatów o błędach nie jest pożądane.

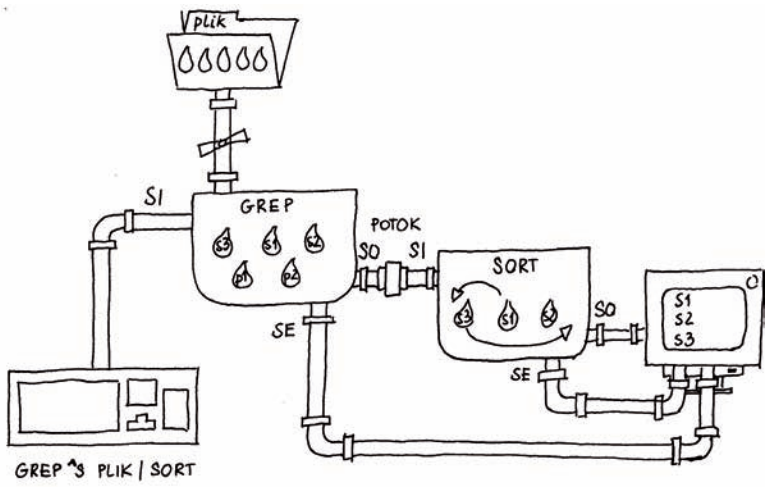
5. `# cat plik4 2>/dev/null`

Aby zablokować wyświetlania komunikatów o błędach na ekranie, należy je przekierować do pliku. Jedną z możliwości jest ich przekierowanie do pliku */dev/null*, w którym wszystkie dane giną. Zauważ, że w poleceniu posłużono się numerycznym deskryptorem standardowego strumienia błędów (cyfra *2*).

Mechanizm potoku

Aby wykonać złożone przetwarzanie danych, niekiedy konieczne staje się uruchomienie wielu procesów, z których kolejny wykorzystuje rezultaty pracy poprzednika. Wówczas dane wyjściowe pierwszego procesu powinny być przekazane do drugiego procesu jako jego dane wejściowe. Tę operację realizuje mechanizm zwany **potokiem** (ang. *pipeline*), oznaczany znakiem | (pionowej linii). Działanie mechanizmu potoku ilustruje rysunek 30.3.

Rysunek 30.3.
Mechanizm potoku



Działanie mechanizmu potoku polega na przekierowaniu strumienia wyjściowego jednego z procesów, np. procesu `grep`, na wejście innego procesu — w pokazanym przykładzie `sort`. Polecenie `grep` odczytuje wskazany plik wiersz po wierszu i pozostawia tylko te z wierszy, które pasują do wzorca — w pokazanym przykładzie zaczynają się od litery *s*. Wybrane wiersze pliku są następnie przekazywane do innego procesu wykonującego polecenie `sort`, które porządkuje wiersze rosnąco lub malejąco. Standardowym wyjściem tego polecenia jest monitor, na którym zostanie wyświetlony rezultat działań obu procesów.

Mechanizm potoku jest powszechnie używany do łączenia poleceń i wykonywania złożonych operacji przetwarzających dane.

W celu wykorzystania mechanizmu potoku wpisz w wierszu poleceń:

1. `# ps -u root`

   Polecenie `ps` z parametrem `-u` wyświetla wszystkie procesy uruchomione przez użytkownika o podanej nazwie, w tym przypadku *root*. Użytkownik ten ma wiele procesów, których lista może nie zmieścić się na jednym ekranie. Aby zobaczyć wszystkie wartości, należy skorzystać z mechanizmu potoku.

2. `# ps -u root | more`

   W podanym poleceniu rezultat wykonania komendy `ps` został przekierowany do polecenia `more`, które umożliwia stronicowanie wyświetlanego tekstu. Zwróć uwagę, że teraz jest widoczny proces *init* o numerze PID równym *1*. Poprzednio był on niewidoczny. Przewiń listę procesów za pomocą klawisza *Spacja*.

3. `# grep ^s plik2 | sort`

   Podane polecenie jest związane z przykładem umieszczonym na rysunku 30.3.

   Polecenie `grep` służy do filtrowania tekstów. Z podanego pliku wejściowego odczytuje kolejne wiersze i na wyjściu pozostawia tylko te, które są zgodne ze wzorcem. Podany wzorzec `^s` oznacza, że pierwszym znakiem wiersza powinna być litera *s*.

   Poprzez mechanizm potoku dane wyjściowe procesu `grep` zostaną przekazane do procesu `sort`. Polecenie `sort` służy do porządkowania wierszy w kolejności alfabetycznej czy numerycznej.

4. `# grep ^s plik2 | sort -r | more`

   Mechanizmu potoku można użyć wielokrotnie, tworząc złożone ciągi operacji przetwarzania danych. Wynik działania polecenia `grep` jest przekazany do programu `sort`, który dzięki opcji wywołania `-r` posortuje dane malejąco. Zauważ, że dane te zostały przekazane do polecenia `more` poprzez kolejny mechanizm potoku.

Najczęściej polecenia do zarządzania procesami przydają się w sytuacji zawieszenia jednego z procesów, gdy zachodzi konieczność jego zatrzymania. Jednak zaawansowane działania, takie jak zarządzanie serwerami czy obliczeniami numerycznymi, wymagają umiejętności nadzorowania stanu procesów czy wykonywania procesów w tle.

Lekcja 31. Przetwarzanie danych tekstowych

Konfiguracja systemu operacyjnego i programów zapisywana jest w plikach tekstowych. Są one przeznaczone do zapisu ustawień programów użytkowych, serwerów usług lub zawierają informacje o aktualnym stanie systemu operacyjnego i środowiskach użytkowników. Dzięki zapisywaniu parametrów działania systemu w plikach jest możliwe odtworzenie tej samej konfiguracji przy każdorazowym uruchomieniu systemu. W trakcie

prac administracyjnych często występuje konieczność znalezienia określonej informacji o systemie i jego konfiguracji zapisanej w pliku tekstowym. Niekiedy zachodzi także konieczność dokonania zmiany zapisanych danych.

Plik tekstowy z danymi konfiguracyjnymi oraz inne pliki tekstowe są zbiorem znaków, które są podzielone na wiersze o różnej długości. Każdy wiersz kończy się specjalnym znakiem. Przetwarzanie plików tekstowych w systemie Linux zawsze opiera się na operacjach na poszczególnych wierszach. Również praca powłoki polega na interpretowaniu poleceń zapisywanych w wierszach. To wiersz jest interpretowany, wykonywany, wyświetlany, zapisywany, a także może być przetwarzany przez programy.

W dystrybucjach systemu Linux znajdują się liczne programy do wykonywania złożonych operacji przetwarzania danych tekstowych; programy te można pogrupować według realizowanych przez nie funkcji:

- wyświetlanie zawartości pliku — `more`, `tail`, `head`, `less`,
- filtrowanie wierszy wg wzorca — `grep`, `egrep`, `fgrep`,
- zliczanie słów i wierszy — `wc`, `nl`, `du`,
- przetwarzanie treści wierszy — `cut`, `tr`, `sed`, `awk`,
- sortowanie wierszy — `sort`, `uniq`,
- łączenie i rozłączanie — `cat`, `join`, `split`, `csplit`,
- porównywanie zawartości plików — `cmp`, `comm`, `diff`.

Niektóre z tych programów mają bardzo duże możliwości przetwarzania plików tekstowych, jednak ich zastosowanie jest niekiedy skomplikowane. Są one wykorzystywane przy pracach administracyjnych.

Poszukiwanie wierszy zawierających wzorzec

Przy wyszukiwaniu informacji w plikach tekstowych często stosuje się program o nazwie `grep`. Program ten filtruje dane, analizując każdy wiersz i pozostawia tylko te z nich, które spełniają warunek zdefiniowany w formie wzorca, podany jako parametr wywołania. Wzorzec może być podany w wierszu poleceń, jak również może być pobierany z pliku, w którym można umieścić wiele różnych wzorców. Wzorzec może być ciągiem znaków, jak również wyrażeniem regularnym, definiującym pewne zbiory danych tekstowych. Istnieje również polecenie `egrep`, które pozwala na stosowanie rozszerzonych wyrażeń regularnych. Wywołanie tego polecenia jest równoznaczne z użyciem w poleceniu `grep` opcji `-E`.

Wzorzec może być wyrażeniem, w którym znaki specjalne mają określone znaczenia, na przykład:

- `ala` — poszukiwany jest podany ciąg znaków;
- `a.a` — znak kropki zastępuje jeden dowolny znak, w tym przypadku poszukiwany jest ciąg zaczynający i kończący się literami *a*, pomiędzy którymi będzie jeden znak (litera albo nawet cyfra);

- `a.*` — znak gwiazdki oznacza powtórzenie znaku poprzedzającego dowolną liczbę razy, w tym przypadku poszukiwane są słowa zawierające ciąg znaków zaczynający się literą *a*;
- `[aeiouy]` — w nawiasie kwadratowym podaje się zbiór poszukiwanych znaków, np. listę samogłosek;
- `[a-zA-Z]` — w nawiasie kwadratowym można podać także zakres poszukiwanych znaków, np. wszystkie wielkie i małe litery;
- `^a` — po znaku ^ podaje się ciąg znaków, który musi wystąpić na początku wiersza, np. literę *a*;
- `a$` — przed znakiem *$* podaje się ciąg znaków, który musi wystąpić na końcu wiersza, np. literę *a*.

Za pomocą wyrażeń regularnych ze znakami specjalnymi można opisać ciągi znaków i ich zbiory. Można stosować także pewne kombinacje poszczególnych znaków specjalnych, co pozwala na wyrażenie bardzo złożonych zależności między słowami czy znakami w pliku tekstowym. Wzorce mogą być stosowane w większości poleceń działających na danych tekstowych. Pozwalają one na wybieranie z dużych plików tekstowych jedynie tych treści, które interesują użytkownika. Najczęściej wzorców używa się w połączeniu z poleceniem `grep`.

Polecenie `grep` posiada kilka opcji wywołania, z których najważniejsze to:

- `-v` — pozostawienie wierszy niepasujących do wzorca,
- `-c` — podanie liczby wierszy pasujących do wzorca,
- `-n` — ponumerowanie znalezionych wierszy,
- `-w` — pozostawienie wierszy, w których są całe słowa, a nie ciągi znaków, pasujące do wzorca,
- `-i` — eliminacja rozróżnienia małych i wielkich liter,
- `-l` — podanie tylko nazw plików zawierających wiersze pasujące do wzorca,
- `-E` — włączenie rozszerzonych wyrażeń regularnych (`egrep`).

Polecenie `grep` pozwala na zmniejszenie ilości danych przekazywanych przez programy kontrolujące system poprzez ich przefiltrowanie i wyświetlenie tylko tych wierszy, które zawierają interesujące dla użytkownika informacje. Poleceniem `grep` można się posłużyć, aby uzyskać czytelny rezultat działania programów przetwarzających dane tekstowe.

W celu poznania możliwości polecenia `grep` wpisz w wierszu poleceń:

1. `# ls /bin`

   Polecenie `ls` wyświetli listę programów systemowych zapisanych w katalogu */bin*. Zauważ, że lista ta jest obszerna, a szukanie w niej jednego z programów może być długotrwałe.

2. `# ls /bin | grep y`

Rezultat działania polecenia `ls` poprzez mechanizm potoku zostanie przekazany jako strumień wejściowy do polecenia `grep`. Jako wzorzec w poleceniu `grep` podano literę *y*, co oznacza, że jedynie polecenia, których nazwa zawiera tę literę, zostaną wyświetlone. W podobny sposób można szukać określonych ciągów znaków i całych wyrazów.

3. `# ls /bin | grep ^[s-w]`

Jako parametr polecenia `grep` można podać zbiory poszukiwanych liter z wykorzystaniem nawiasów kwadratowych. Dodatkowy znak specjalny `^` w wyrażeniach regularnych oznacza początek wiersza. Jako wynik polecenia zostaną wyświetlone wszystkie nazwy plików z katalogu */bin* które zaczynają się na litery *s*, *t* i kolejne do *w*.

4. `# ls /bin | grep -i [ouy]$`

Znak specjalny *$* w wyrażeniu regularnym oznacza koniec wiersza. W nawiasie kwadratowym podano zbiór liter, które wchodzą w skład wzorca. Rezultatem wydanego polecenia będzie wyświetlenie plików, których nazwa kończy się jedną z podanych samogłosek: *o*, *u* lub *y*. Wywołanie polecenia `grep` z opcją `-i` powoduje, że podczas poszukiwania wielkie i małe litery nie są rozróżniane. Tak samo zadziała polecenie bez opcji, ale ze wzorcem `[ouyOUY]`.

5. `# ls -l /dev | wc -l`

Polecenie `wc` w zależności od zastosowanej opcji służy do zliczania poszczególnych elementów zawartości pliku. Opcja `-l` (ang. *line*) oznacza zliczanie wierszy, `-w` (ang. *word*) — słów, a opcja `-c` (ang. *character*) — znaków. Danymi wejściowymi może być zawartość pliku tekstowego, ale także rezultat wykonanego polecenia przekazany na standardowe wejście procesu `wc`. Ponieważ polecenie `ls` z opcją `-l` wyświetla dane o każdym pliku i katalogu w oddzielnym wierszu, przekazanie rezultatu do polecenia `wc` zliczającego wiersze pozwoli na podanie liczby plików i katalogów w danej lokalizacji. Zwróć uwagę, że katalog ten zawiera dużą liczbę elementów.

6. `# ls -l /dev | grep ^d | wc -l`

Polecenie `ls` wywołane z opcją `-l` wyświetla dla każdego elementu w podanej lokalizacji pełną o nim informację, w której znajduje się także typ pliku. Typ jest oznaczany jedną literą umieszczoną na początku wiersza dotyczącego danego pliku, np. litera *d* oznacza katalogi. Wykorzystując tę zasadę, można za pomocą polecenia `grep` wyświetlić jedynie wiersze zawierające wybrany typ plików. W rezultacie wydanego polecenia zostanie podana liczba katalogów w wybranej lokalizacji.

Przetwarzanie danych tekstowych

Dane tekstowe zawarte w plikach mogą być wyświetlane za pomocą takich poleceń, jak `cat`, `less` czy `more`. Jednocześnie istnieje możliwość ręcznej edycji danych w edytorach tekstu, takich jak `nano` czy `vim`. Jednak niekiedy zachodzi konieczność automatycznego przetwarzania plików, gdy ręczne ich przeszukiwanie i poprawianie byłoby

nieefektywne. Czasem modyfikacje polegają na zmianie jednej litery tekstu w wielu wierszach pliku. Przetworzenie dużych plików przez użytkownika może być pracochłonne, a w niektórych przypadkach — wręcz niewykonalne. Zmiany tego rodzaju mogą zostać wykonane automatycznie przez przeznaczone do tego celu programy, takie jak `cut`, `tr` czy `sed`.

W niektórych przypadkach plik tekstowy posiada własną wewnętrzną strukturę, dzieli się na wiersze, słowa i kolumny. Wiersze są rozpoznawane dzięki kończącemu je specjalnemu znakowi końca wiersza. Słowo to ciąg znaków niepodzielony spacją lub innym znakiem specjalnym. Wyróżnienie kolumn występuje zazwyczaj jedynie w plikach generowanych automatycznie przez określone programy i wówczas poszczególne kolumny zawierają zawsze te same informacje i znajdują się w wierszu w tej samej kolejności, a oddzielone są od siebie znakiem specjalnym. Taką postać często mają pliki z zapisem ustawień systemu operacyjnego. Również polecenia systemowe często zwracają dane w postaci uporządkowanej w wiersze i kolumny. Gdy znana jest struktura danych, programy do przetwarzania mogą dokonywać zmian automatycznie. Dokonane w strumieniu wejściowym zmiany mogą być zapisane do plików wynikowych. Całość działania programu może przebiegać bez ingerencji użytkownika i bez wyświetlania jakichkolwiek komunikatów na ekranie.

W celu przetworzenia wybranych danych tekstowych wpisz w wierszu poleceń:

1. `# cat /etc/passwd`

   Program `cat` odczyta zawartość pliku */etc/passwd* i wyświetli ją na monitorze. Plik */etc/passwd* pełni bardzo ważną rolę w pracy systemu operacyjnego, ponieważ zawiera informacje o użytkownikach. W każdym wierszu znajdują się kolejno: nazwa użytkownika, jego hasło, numer UID, numer GID, imię i nazwisko, katalog domowy oraz zdefiniowana powłoka. Zauważ, że plik ten ma ściśle określoną strukturę, a poszczególne kolumny oddzielone są znakiem dwukropka. Zwróć uwagę, że zamiast hasła wyświetlany jest znak *x*, który oznacza, że hasła zostały zaszyfrowane i zapisane w oddzielnym pliku */etc/shadow*. Taki sposób zapisu haseł jest obecnie ze względów bezpieczeństwa powszechnie stosowany w systemach operacyjnych Linux.

2. `# cat /etc/passwd | cut -d ":" -f 1,6`

   Jeżeli chcesz wyświetlić wybrane informacje, np. tylko nazwę użytkownika i jego katalog domowy, możesz posłużyć się poleceniem `cut`. Program `cut` pozwala na wycięcie pewnego fragmentu z wierszy tekstu. Potrafi operować na wierszach i kolumnach analizowanego tekstu. W poleceniu `cut` za pomocą opcji `-d` definiuje się, jaki znak oddziela poszczególne kolumny. W przypadku pliku */etc/passwd* jest to znak dwukropka. Opcja `-f` pozwala na wybranie kolumn, które mają być wyświetlane. Zauważ, że w wydanym poleceniu znajduje się wywołanie dwóch programów o nazwach `cut` i `cat`. Mimo podobnej pisowni oba programy realizują zupełnie inne zadania. Odczytywana za pomocą polecenia `cat` zawartość pliku */etc/passwd* jest przekazywana za pomocą mechanizmu potoku na wejście programu `cut`. W rezultacie zostaną wyświetlone jedynie kolumny pierwsza i szósta, w których znajdują się nazwa użytkownika i nazwa

jego katalogu domowego. Polecenie `cut` posiada liczne opcje, które umożliwiają definiowanie skomplikowanych warunków, określających, jakie dane mają zostać wyświetlone, a jakie wycięte.

3. `# cat /etc/passwd | cut -d ":" -f 1,6 | tr ":" " "`

   Dane zapisane w plikach systemowych mają często mało czytelną postać. Dlatego oprócz wycinania niepotrzebnych kolumn można zamienić jeden znak specjalny na inny. W rezultacie wykonania polecenia znak dwukropka zostanie zamieniony na znak spacji za pomocą polecenia `tr`.

4. `# cat /etc/passwd | cut -d ":" -f 1,6 | tr "[a-z]" "[A-Z]"`

   Polecenie `tr` pozwala również na zamianę grup znaków. W tym przypadku wszystkie małe litery zostaną zamienione na wielkie. Zauważ, że dokonywane zmiany są widoczne w postaci informacji wyświetlanych na ekranie. Zawartość pliku */etc/passwd* nie ulega zmianie. Aby zapisać zmiany, konieczne jest jawne przekierowanie danych do pliku o określonej nazwie. Polecenie `tr` ma ograniczone możliwości — zamienia jedynie pojedyncze znaki. Aby zamienić cały ciągów znaków, konieczne jest posłużenie się poleceniem `sed`.

5. `# cat /etc/passwd | cut -d ":" -f 1,6 | sed "s/:/ ma katalog domowy /"`

   Program `sed` umożliwia zaawansowane przetwarzanie wierszy tekstu z wykorzystaniem wyrażeń regularnych. Działanie programu polega na pobieraniu wierszy jeden po drugim i przetwarzaniu jego treści zgodnie ze zdefiniowanymi operacjami i wzorcami. Jedną z operacji realizowanych przez program jest zamiana pewnego ciągu znaków na inny. Jest to możliwe dzięki operacji zamiany oznaczonej literą `s`, której użycie wymaga podania polecenia w postaci: `s/`*`ciąg początkowy/ciąg docelowy/`*. W rezultacie wykonania polecenia znak dwukropka oddzielający kolumny w pliku */etc/passwd* zostanie zastąpiony zwrotem *ma katalog domowy*.

6. `# cat /etc/passwd | cut -d ":" -f 1,6 | sed -e "s/^/Użytkownik /" \`
   `-e "s/:/ ma katalog domowy /"`

   Gdy w jednym wierszu danych konieczne jest dokonanie kilku zamian, należy posłużyć się opcją `-e` polecenia `sed`. Może być ona podawana wielokrotnie, a po niej należy zdefiniować rodzaj wykonywanej operacji. W podanym przykładzie została ona użyta dwukrotnie: do zamiany znaku początku wiersza (^) na słowo *Użytkownik* oraz do zamiany znaku dwukropka na napis *ma katalog domowy*. W ten sposób z nieczytelnych danych zawartych w pliku można uzyskać nawet kompletne zdania opisujące parametry działania systemu operacyjnego. Zwróć uwagę na znak `\` (ang. *backslash*), który pozwala na podawanie jednego długiego polecenia w kilku wierszach. Oprócz zamiany ciągów znaków polecenie `sed` może realizować wiele innych operacji przetwarzania danych, ma bogaty zestaw opcji, a nawet własny język programowania.

Kontrola danych tekstowych

Dane tekstowe często są zapisywane albo wyświetlane w postaci nieuporządkowanej. Aby zwiększyć czytelność tekstu, można je posortować za pomocą polecenia `sort` albo usunąć występujące powtórzenia za pomocą polecenia `uniq`. Jednocześnie niekiedy użytkownika nie interesuje zawartość pliku, a jedynie dane liczbowe ją określające. Można je uzyskać, stosując wyspecjalizowane programy, takie jak `wc`, `du`, `nl` czy `md5sum`. Operacje filtrowania, sortowania czy zliczania danych pozwalają na uzyskanie istotnych informacji z dużego zbioru danych.

W celu poznania możliwości programów do kontroli danych tekstowych wpisz w wierszu poleceń:

1. `# wc /etc/passwd; md5sum /etc/passwd`

   Polecenia `wc` i `md5sum` pozwalają na wyświetlenie liczbowych danych o zawartości pliku tekstowego. Polecenie `wc` pozwala na podanie liczby wierszy, słów i znaków. Wywołując polecenie `wc` z opcjami, można określić, jakie elementy mają zostać wyświetlone. Polecenie `md5sum` podaje tzw. sumę kontrolną pliku. Jest to liczba, którą oblicza się na podstawie zawartości pliku. Pozwala ona na kontrolę, czy plik został zmodyfikowany. Zapis plików i ich sum kontrolnych może być wykorzystywany dla zapewniania bezpieczeństwa ważnych danych.

2. `# du --max-depth=1 /etc`

   Polecenie `du` służy do wyświetlania informacji o łącznej wielkości wszystkich plików i podkatalogów w podanym katalogu. Domyślnie wyświetlana jest liczba bajtów, jednak za pomocą opcji programu można zmienić jednostkę wyświetlania. W poleceniu zastosowano jedną z opcji programu `du` o nazwie `--max-depth`. Zwróć uwagę na dwa znaki minus, które stosuje się w przypadku, gdy opcja składa się z więcej niż jednej litery. Użyta opcja z parametrem *1* określa, że mają być wyświetlone jedynie informacje o katalogach, bez dalszego wyszczególniania ich podkatalogów. Oznacza to, że będzie wyświetlany tylko jeden poziom zagłębienia w drzewie katalogu */etc*. Przeanalizuj rezultat działania polecenia i zwróć uwagę na dużą ilość nieuporządkowanych danych, które można przeglądać, wykorzystując klawisze *Shift*+*PgUp* i *Shift*+*PgDn*.

3. `# du --max-depth=1 /etc | sort -k 2`

   Poprzez przekierowanie rezultatu działania polecenia `du` do programu `sort` można uporządkować uzyskane wyniki. W tym przypadku poprzez opcję `-k` wskazano, po której kolumnie ma zostać przeprowadzone sortowanie. Wyniki zostaną uporządkowane alfabetycznie według nazw katalogów.

4. `# du --max-depth=1 /etc | sort -k 1 -g -r`

   Ponieważ polecenie `du` podaje informacje o wielkości katalogów, rezultat jego działania można wykorzystać do poszukiwania katalogów o największej wielkości. W tym celu należy posortować dane według wartości pierwszej kolumny. Opcja `-g` polecenia `sort` określa, że dane w kolumnie mają być

traktowane jak liczby, a nie jak ciągi znaków. Opcja `-r` określa malejący porządek sortowania i oznacza, że największe pliki będą umieszczane na początku listy.

5. `# ps -ef | cut -d " " -f 1`

   Polecenie `ps` z opcjami `-ef` wyświetla pełne informacje o uruchomionych w systemie procesach. Danych jest dużo, a nie wszystkie są interesujące dla użytkownika. Z tych względów zastosowano polecenie `cut` do usunięcia wszystkich kolumn oprócz nazwy użytkownika. Na wyświetlonej liście mogą znajdować się nie tylko zalogowani użytkownicy, lecz także programy, które pracują jako serwery usług. Zauważ, że ponieważ użytkownik *root* miał wiele uruchomionych procesów, nazwa jego konta pojawia się wielokrotnie.

6. `# ps -ef | cut -d " " -f 1 | uniq -c`

   Polecenie `uniq` służy do eliminowania kolejnych wierszy o takiej samej zawartości, a wywołane z opcją `-c` pozwala je dodatkowo policzyć. Rezultatem wydanego polecenia jest informacja o liczbie uruchomionych procesów przez poszczególnych użytkowników systemu operacyjnego. Zauważ, że informacja o użytkowniku *root* pojawia się kilkukrotnie. Jest to związane z pracą polecenia `uniq`, które grupuje jedynie następujące bezpośrednio po sobie wiersze o takiej samej zawartości.

7. `# ps -ef | cut -d " " -f 1 | sort | uniq -c`

   Wiersz uzupełniono o polecenie `sort`, które przed zastosowaniem polecenia `uniq` uporządkowało nazwy użytkowników alfabetycznie. Dzięki temu podano sumaryczną liczbę uruchomionych procesów dla użytkownika *root*.

Zarządzanie systemem operacyjnym niejednokrotnie wymaga od użytkownika operowania na plikach konfiguracyjnych systemu. Należą do nich między innymi pliki:

- z danymi użytkowników — */etc/passwd*, */etc/shadow*;
- z logami systemowymi, np. */var/log/syslog*, */var/log/messages*;
- z informacjami o zasobach komputera, np. */proc/meminfo*, */proc/procinfo*;
- konfiguracji sieci Internet — */etc/network/interfaces*;
- definiujące środowisko pracy użytkowników, np. */etc/profile*, */etc/bash.bashrc*.

Są to pliki tekstowe o często znacznych rozmiarach, a ręczne ich przetwarzanie byłoby kłopotliwe. Z tych względów w system operacyjny Linux wbudowano szereg zaawansowanych programów ułatwiających pracę z dużymi plikami tekstowymi. Warto zauważyć, że tak samo jak pliki tekstowe przetwarzane są rezultaty wykonanych poleceń, np. wyświetlenia zawartości katalogu czy listy uruchomionych procesów. Programy powłoki przetwarzają wszystkie dane tekstowe w taki sam sposób — wiersz po wierszu. Również rezultaty działania programów są wyświetlane w postaci wierszy. Jest to cechą charakterystyczną środowisk tekstowych w systemach operacyjnych Linux i Unix.

Lekcja 32. Definiowanie środowiska pracy użytkownika

Po poprawnym zalogowaniu się użytkownika uruchamiane jest jego środowisko pracy. Na środowisko pracy składa się domyślny program powłoki wraz z jego ustawieniami — sposobem zapamiętywania historii poleceń, rodzajem wyświetlanego znaku zachęty, domyślnymi prawami dostępu do nowo tworzonych plików, ścieżkami dostępu do katalogów z poleceniami systemowymi i innymi parametrami. Elementy środowiska są zapisywane w zmiennych, których nazwy zwyczajowo są pisane wielkimi literami. Wartości zmiennych mogą być wyświetlane, a także zmieniane.

Proces budowy środowiska pracy użytkownika przez system operacyjny składa się z dwóch kroków. W pierwszym wartości zmiennych środowiska są odczytywane z plików wspólnych dla wszystkich użytkowników — */etc/profile* oraz */etc/bash.bashrc*. W drugim kroku sprawdzane są indywidualne ustawienia użytkownika zapisane w jego katalogu domowym w plikach *.profile* oraz *.bashrc*. Indywidualne ustawienia zamieniają (nadpisują) wcześniej przypisane domyślne wartości zmiennych. Mogą także definiować nowe zmienne. Jeżeli wśród indywidualnych ustawień nie ma podanych wartości zmiennych — pozostawiane są ustawienia domyślne. Takie rozwiązanie umożliwia użytkownikom rozpoczęcie pracy bez konieczności konfiguracji środowiska. Domyślne wartości zmiennych są wystarczające dla większości wykonywanych przez użytkowników prac. Jednak niekiedy zachodzi potrzeba zmiany wybranych ustawień środowiska, które można w ten sposób dostosować do własnego stylu pracy.

Definicja synonimu polecenia

Jednym za sposobów tworzenia własnego środowiska pracy jest stworzenie własnych synonimów poleceń wraz z ich parametrami wywołania. Przypomina to trochę uczenie małego dziecka nowych słów na podstawie tych, które już zna. Pewnemu wierszowi polecenia nadaje się osobisty skrót, zwany **aliasem**, który można wielokrotnie wykorzystywać. Do definiowania tego skrótu służy polecenie powłoki `alias`. Przypisuje ono słowu wykonywalny wiersz poleceń, złożony z polecenia wraz z parametrami wywołania. Powłoka po otrzymaniu polecenia, będącego zdefiniowanym synonimem, wykona przypisany do niego wiersz poleceń.

Aby stworzyć synonim polecenia, wpisz w wierszu poleceń:

1. `# alias nowehaslo='passwd'`

   Program `alias` służy do definiowania własnego synonimu polecenia. Jako parametr podaje się nazwę synonimu oraz po znaku równości polecenie, które ma zostać wykonane przy posłużeniu się tą nazwą. Zwróć uwagę, że przed znakiem równości i po nim nie ma znaku spacji, a polecenie jest ujęte w znaki apostrofu, które są konieczne, gdy w poleceniu znajdują się spacje. W rezultacie wykonania polecenia zostanie zdefiniowany alias `nowehaslo`, będący synonimem polecenia `passwd`.

2. `# nowehaslo`

Zmień hasło, korzystając z nowo utworzonego aliasu. Po wydaniu polecenia `nowehaslo` powłoka rozpozna je jako alias polecenia `passwd`, które zostanie wykonane. Możesz dokonać zmiany hasła tak samo jak przy wykorzystaniu polecenia `passwd`.

3. `# alias cojest ='ps -u root | grep tty1'`

Mechanizm aliasów jest szczególnie korzystny, gdy często powtarzane polecenie jest długie. Dla złożonego polecenia wyświetlającego procesy użytkownika *root* uruchomione na pierwszym terminalu można zdefiniować krótszą nazwę, np. `cojest`.

4. `# cojest`

Uruchom polecenie wyświetlające listę procesów za pomocą zdefiniowanego wcześniej aliasu. Warto pamiętać, że definiowanie długich nazw aliasów nie jest korzystne, ponieważ wymaga długiego wpisywania. Jednocześnie nazwa powinna logicznie odpowiadać wykonywanej operacji. Podobnie do nazw poleceń systemowych nazwy aliasów są uzupełniane po wpisaniu pierwszych liter i naciśnięciu klawisza *Tab*.

5. `# alias`

Polecenie `alias` wydane bez parametrów wyświetli listę wszystkich zdefiniowanych synonimów. Zauważ, że oprócz zdefiniowanych wcześniej własnych aliasów na liście znajdują się także aliasy zdefiniowane dla wszystkich użytkowników. Jednym z najczęściej występujących w systemie Linux aliasów jest `ll`, który oznacza polecenie `ls` z opcją `-l`.

6. `# unalias nowehaslo`

Polecenie `unalias` służy do usuwania wcześniej zdefiniowanych synonimów.

Zmienne powłoki

Środowisko pracy użytkownika jest definiowane przez zmienne systemowe zapisane w plikach konfiguracyjnych, a po uruchomieniu systemu w pamięci operacyjnej komputera. Ważniejsze zmienne określają bieżącego użytkownika (`USER`), bieżący katalog (`PWD`), katalog domowy (`HOME`), używany język (`LANG`) czy używany program powłoki (`SHELL`). Określane są także lokalizacje i rozmiary ważnych plików, takich jak plik historii poleceń użytkownika (zmienne `HISTFILE` oraz `HISTSIZE`). Jedną z najczęściej modyfikowanych zmiennych jest `PATH`, określająca lokalizacje, w których powłoka poszukuje programów i poleceń. W zmiennej `PATH` zazwyczaj znajduje się wiele lokalizacji, z których najważniejsze to */bin*, */sbin*, */usr/bin* oraz */usr/sbin*. W tych katalogach znajduje się większość wykonywalnych plików umożliwiających uruchamianie poleceń systemowych. Programy umieszczone w tych katalogach mogą być uruchomione z dowolnego miejsca w drzewie katalogów, bez podawania ich lokalizacji. Dzięki zmiennej `PATH` użytkownik może wydawać polecenia, nie zdając sobie sprawy z tego, że uruchamia plik wykonywalny zapisany głęboko w drzewie katalogów.

W celu wyświetlenia wartości zmiennych systemowych wpisz w wierszu poleceń:

1. `# echo $PATH`

   Polecenie `echo` może służyć do wyświetlenia wartości zmiennej środowiska podanej jako parametr wywołania. Zauważ, że nazwa zmiennej została poprzedzona znakiem *$*, który oznacza, że ma być wypisana jej wartość. W ten sposób można wyświetlić wartość każdej ze zmiennych systemowych. Zmienna `PATH` definiuje, jakie katalogi są przeglądane przez powłokę w momencie wydanie polecenia przez użytkownika. Domyślna wartość zmiennej `PATH` definiowana jest w globalnym pliku konfiguracyjnym */etc/profile*, który definiuje środowisko pracy dla wszystkich użytkowników.

2. `# set | more`

   Polecenie `set` wyświetla wartości wszystkich zmiennych systemowych. Zauważ, że jest ich wiele, a przypisywane wartości mogą być długimi ciągami znaków.

3. `# PATH=$PATH:.`

   Zmień wartość zmiennej `PATH`, dodając do niej katalog bieżący. Zauważ, że jako nowa wartość zmiennej została podana poprzednia wartość zmiennej oznaczona `$PATH` i dodatkowo po znaku dwukropka katalog bieżący oznaczony znakiem kropki. Znak dwukropka został użyty, ponieważ poszczególne katalogi oddzielane są w zmiennej `PATH` tym znakiem. Zaleca się, aby nowe ścieżki dostępu dodawać na końcu zmiennej `PATH` i nie usuwać standardowych wartości, ponieważ może to doprowadzić do niekorzystnych zmian w środowisku pracy użytkownika. Dodanie katalogu bieżącego do zmiennej `PATH` pozwala na uruchamianie skryptów i programów znajdujących się w katalogu bieżącym.

4. `# echo $PATH`

   Wyświetl ponownie wartość zmiennej `PATH` i zauważ, że została ona rozszerzona o katalog bieżący. W podobny sposób można zmieniać wartość każdej ze zmiennych systemowych.

Definiowanie znaku zachęty

Powłoka *bash* w chwili uruchomienia wyświetla na ekranie pewien ciąg znaków ją identyfikujący, po którym wpisywane są interpretowane polecenia. Ten ciąg znaków nazywany jest znakiem zachęty. Domyślnie w dystrybucji cdliunx.pl składa się on z nazwy użytkownika, znaku @, nazwy komputera, znaku dwukropka i nazwy bieżącego katalogu. Znak zachęty zakończony jest znakiem # dla użytkownika *root* albo znakiem *$* dla pozostałych użytkowników. Liczba i rodzaj elementów tworzących znak zachęty mogą być dowolne i są definiowane w zmiennej środowiska o nazwie `PS1`.

W celu zmiany znaku zachęty dla powłoki wpisz w wierszu poleceń:

1. `# echo $PS1`

   Wyświetlona zawartość zmiennej `PS1` definiuje kolejne składowe znaku zachęty, w tym specjalne symbole interpretowane przez powłokę, która

podstawia w ich miejsce określone dane. Symbol *\u* oznacza nazwę bieżącego użytkownika, po niej następuje znak @, ciąg *\h* jest zastępowany nazwą komputera, a ciąg *\w* nazwą bieżącego katalogu użytkownika.

2. `# PS1="*>"`

   Przypisując do zmiennej `PS1` w wierszu poleceń dowolny ciąg znaków, można dokonać chwilowej zmiany znaku zachęty. W tym przypadku zdefiniowany został nowy znak zachęty, składający się jedynie ze znaku gwiazdki i znaku większości. Zauważ zmianę znaku zachęty po naciśnięciu klawisza *Enter*. Zmiana znaku zachęty dotyczy jedynie bieżącego terminalu i programu powłoki. Po zalogowaniu się na innym terminalu możesz zauważyć, że znak zachęty jest zgodny z początkowymi ustawieniami zapisanymi w plikach konfiguracyjnych.

3. `# PS1="\u:\w:\d# "`

   Zmień ponownie znak zachęty, korzystając z symboli, które związane są ze zmiennymi systemowymi. Ciąg znaków *\u* zostanie zastąpiony nazwą użytkownika, *\w* — nazwą bieżącego katalogu, a *\d* — bieżącą datą. Znak zachęty będzie się kończył znakiem #. Definicja znaku zachęty może zawierać wiele różnych elementów, jednak, podobnie jak z ikonami na pulpicie, należy zachować umiar.

Zmiany w środowisku pracy

Stworzony za pomocą polecenia `alias` synonim czy zdefiniowany za pomocą zmiennej `PS1` nowy znak zachęty będą interpretowane przez powłokę jedynie do momentu jej zamknięcia. Przy kolejnym zalogowaniu użytkownika system ponownie skorzysta z ustawień domyślnych zapisanych w plikach konfiguracyjnych. W przypadku gdy użytkownik chce trwale zmienić środowisko pracy, powinien dokonać modyfikacji pliku określającego to środowisko. Dla powłoki *bash* plikiem takim jest *.bashrc* znajdujący się w katalogu domowym użytkownika. Znak kropki przed jego nazwą symbolizuje plik ukryty, co oznacza, że nie jest on widoczny przy typowym wyświetlaniu zawartości katalogu. Pliki ukryte wyświetla opcja `-a` polecenia `ls`.

W celu dokonania trwałych zmian w środowisku pracy wykonaj następujące kroki:

1. Za pomocą polecenia `more` wyświetl zawartość pliku *.bashrc* znajdującego się w katalogu domowym użytkownika *root*. Zapoznaj się z jego zawartością. Zwróć uwagę na zapisy o zdefiniowanych aliasach. Wiersze, które zaczynają się znakiem #, są komentarzem i nie są wykonywane. W pliku *.bashrc* zapisywane są ustawienia użytkownika, odczytywane przy jego logowaniu. W pliku tym użytkownik może zapisać przez siebie zdefiniowane aliasy, wartości zmiennych czy inne preferowane ustawienia.

2. Otwórz w edytorze tekstowym `nano` plik *.bashrc*. Znajdź wiersz z definicją znaku zachęty i zmień wartość zmiennej `PS1`. Zmiana wartości zmiennej zapisana w pliku będzie widoczna także przy kolejnych logowaniach na innym terminalu. Zapisz plik i przejdź na terminal drugi (klawisze *Alt+F2*). Zaloguj się i zauważ, że znak zachęty został odczytany z pliku. Wróć do edycji pliku z ustawieniami na terminalu pierwszym.

3. Usuń znak komentarza # sprzed wiersza:

```
# alias rm='rm -i'
```

Znak komentarza oznacza, że polecenie zawarte w danym wierszu nie jest wykonywane. Usunięcie znaku # spowoduje, że definicja aliasu będzie aktywna. Od tej chwili wykonanie polecenia `rm` zostanie zastąpione poleceniem `rm` z opcją `-i`, które spowoduje, że powłoka będzie zawsze pytała o potwierdzenie usunięcia pliku. Dzięki takiemu ustawieniu można uniknąć przypadkowego skasowania pliku.

4. Dopisz na końcu pliku *.bashrc* definicję nowego synonimu:

```
alias nowehaslo='passwd'
```

Aliasy, które są zapisane trwale w pliku *.bashrc* będą widoczne przy kolejnym logowaniu. Zapisz zawartość pliku z dokonanymi zmianami i wyjdź z edytora. Jeżeli bezpośrednio po modyfikacji pliku spróbujesz użyć zdefiniowanego aliasu, nie będzie on działał, ponieważ plik *.bashrc* jest czytany w chwili logowania do systemu. Zmiany będą widoczne po zalogowaniu w nowym terminalu.

5. Ponownie otwórz w edytorze `nano` plik *.bashrc*. Zwróć uwagę na wiersz zawierający polecenie:

```
umask 022
```

Polecenie `umask` służy do odbierania użytkownikom wybranych praw do nowo tworzonych plików. Podobnie jak w przypadku polecenia `chmod` prawa określa się w formie liczbowej. Domyślne ustawienie zapisane w pliku *.bashrc* oznacza, że odbierane jest prawo do pisania dla grupy i pozostałych użytkowników. Oznacza to, że standardowo powłoka, tworząc plik, nadaje wszystkim użytkownikom prawo do czytania. Prawa do katalogów typowo obejmują czytanie i wykonywanie, które oznacza możliwość wyświetlania zawartości katalogu. Jedynie właściciel ma prawo do modyfikacji plików i katalogów. Zmień parametr polecenia `umask` na `027`. W ten sposób do nowo tworzonych plików pozostali użytkownicy nie będą mieli nadanych żadnych praw.

6. Zapisz plik *.bashrc* i wyjdź z edytora. Wyloguj się z systemu za pomocą polecenia `exit`. Zaloguj się ponownie do systemu.

7. Sprawdź za pomocą poleceń `set` oraz `alias`, że zapisy dokonane w pliku *.bashrc* powodują trwałe zmiany w środowisku pracy.

Dla każdego użytkownika systemu operacyjnego jego środowisko pracy może być w pewnym zakresie inne. Wynika to z indywidualnego sposobu pracy i zadań, jakie użytkownik ma do wykonania. Początkowo użytkownik korzysta z ustawień domyślnych, ponieważ są one wystarczające do typowych zadań. Czasem zachodzi jednak konieczność dopasowania środowiska do zmieniających się potrzeb użytkownika. Jedną z możliwości konfiguracji jest ustawianie zmiennych systemowych, czego można dokonać w bieżącej powłoce. Zmiany mogą także zostać trwale zapisane w indywidualnych plikach konfiguracyjnych. Pracując w dystrybucji cdlinux.pl uruchamianej z płyty CD,

warto pamiętać, że zmiany dokonane w plikach katalogu bieżącego zostaną utracone po wyłączeniu komputera. Daje to możliwość eksperymentowania z ustawieniami środowiska. Jednak aby zmiany były stałe, należy zapisać plik z ustawieniami na nośniku trwałym. Można także dokonywać zapisu tego pliku za pośrednictwem poznanych już mechanizmów zarządzania konfiguracją. Innym sposobem usprawnienia pracy użytkowników jest tworzenie i uruchamianie skryptów przeznaczonych do zarządzania systemem operacyjnym.