2. Prawa dostępu plików i dowiązania

2.1. Prawa dostępu plików

W systemach klasy UNIX dostęp do plików i katalogów (oraz innych obiektów, które obsługiwane są tak, jakby były plikami) kontrolowany jest przez tak zwane **prawa dostępu** (ang. *access rights* ewentualnie *file permissions*), które regulują zasady, na jakich użytkownicy mogą korzystać z tych zasobów. Wyróżnia się trzy podstawowe rodzaje dostępów:

- odczyt – oznaczane jako **r** (od ang. *read*)
- zapis – oznaczane jako **w** (od ang. *write*)
- wykonanie – oznaczane jako **x** (od ang. *execute*)

Każdy z powyższych typów dostępów jest dopuszczany bądź wykluczany niezależnie dla:

- użytkownika (ang. *user*), który jest właścicielem pliku lub katalogu (domyślnie właścicielem jest ten użytkownik, który utworzył dany plik lub katalog)
- użytkowników, którzy należą do tej samej grupy (ang. *group*), do której przynależy plik lub katalog (jest to tak zwana *grupa właścicielska*)
- pozostałych (ang. *other*) użytkowników

Interpretacja praw dostępu w przypadku zwykłych plików (tzn. plików niebędących katalogami) jest następująca:

– **r** – umożliwia **odczytanie** zawartości pliku, na przykład w celu wyświetlenia jego zawartości w konsoli albo skopiowania pliku pod inną nazwę
– **w** – umożliwia **zapis** do pliku oraz jego **usunięcie**, uprawnieniem tym jest na przykład regulowana możliwość edytowania zawartości pliku ale już nie przemianowanie go
– **x** – umożliwia **uruchomienie** pliku, gdy zawiera on kod wykonywalny, niezależnie od tego, czy zawartość jest kodem binarnym, czy też tekstem skryptu.

Znaczenie praw zmienia się jednak radykalnie, kiedy odnoszą się do katalogu:
– **r** – umożliwia odczytanie metadanych plików znajdujących się w katalogu (co pozwala na przykład skutecznie wykonać polecenie **`ls`** w tym katalogu)
– **x** – umożliwia uczynienie katalogu katalogiem bieżącym, czyli daje możność wykonania takiego polecenia **`cd`**, w wyniku którego przemieścimy się do wnętrza takiego katalogu
– **w** – umożliwia tworzenie, usuwanie i przemianowywanie plików znajdujących się w katalogu, ale tylko wtedy, gdy jednocześnie posiada się do tego katalogu uprawnienie **x**; bez uprawnienia **x** uprawnienie **w** jest pozbawione znaczenia

Podstawowym sposobem zasięgania informacji o prawach plików i katalogów jest użycie polecenia **`ls`** z przełącznikiem **`-l`** – oto przykład wyjścia wyprowadzanego przez to polecenie użytego w katalogu, w którym istnieje już plik *abc.txt*:

```
$ ls -l
drwx------ 15 adam students 4096 lip  6 13:27 ./
drwxr-xr-x 54 adam students 4096 lip  6 11:20 ../
-rwxr--r-x  2 adam students 4096 cze 23 13:32 abc.txt
```

Pierwszy znak maski praw wyświetlanej przez polecenie **`ls -l`** używany jest do uwidocznienia rodzaju pliku, i tak:
`-` sygnalizuje brak szczególnych własności prezentowanego pliku, a więc jest to **plik zwykły**,
`d` oznacza, że prezentowany plik to **katalog** (od ang. *directory*).

Informacja o prawach wyświetlana na kolejnych dziewięciu znakach według następującego schematu:

user			*group*			*other*		
r	w	x	r	w	x	r	w	x

Wystąpienie na wskazanej pozycji jednego z powyższych znaków oznacza istnienie danego prawa, znak „–” oznacza jego brak.

Zatem dla pliku *abc.txt* określone są następujące prawa:
- właściciel ma komplet praw do swojego pliku (*rwx*)
- dla członków grupy *students* dostępny jest tylko odczyt (*r–*)
- pozostałym użytkownikom zezwala się na odczyt i wykonanie (*r-x*)

Prawami dostępu można także operować stosując notację numeryczną, w której każde z uprawnień traktowane jest jako **bit** (istnienie prawa jest *jedynką*, brak prawa *zerem*), a uzyskaną w ten sposób wartość binarną prezentuje się jako liczbę **ósemkową**. Skoro więc komplet praw opisywany jest dziewięcioma bitami, równoważna postać ósemkowa składać się będzie z trzech cyfr. Każda z nich uwidacznia prawa jednego rodzaju użytkowników (u, g, o) i tworzona jest jako suma arytmetyczna następujących wartości:
- 4 – uprawnienie **r**
- 2 – uprawnienie **w**
- 1 – uprawnienie **x**

Wynika stąd, że prawa pliku *abc.txt* zapisuje się w tej notacji jako **`745`**, gdyż:
- user: **`rwx`** $\Rightarrow 4 + 2 + 1 = 7$
- group: **`r--`** $\Rightarrow 4 + 0 + 0 = 4$
- others: **`r-x`** $\Rightarrow 4 + 0 + 1 = 5$

Należy pamiętać, że wpływ na to, jak i przez kogo może być używany dany plik, ma również to, do kogo plik należy i do jakiej grupy właścicielskiej jest przypisany. System Linux udostępnia zestaw poleceń umożliwiający efektywne manipulowanie każdą z tych danych. Są to:
- **`chmod`** (ang. *change mode*) do zmiany praw dostępu do pliku
- **`chown`** (ang. *change owner*) do zmiany właściciela pliku (chociaż należy zaznaczyć, że polecenie to pozwala również zmieniać grupę pliku)
- **`chgrp`** (ang. *change group*) do zmiany grupy właścicielskiej pliku

2.1.1. Polecenie `chmod`

`chmod` ***`[przełączniki...] uprawnienia nazwa_pliku...`***

Polecenie **`chmod`** zmienia prawa dostępu w sposób wskazany pierwszym argumentem wywołania na rzecz plików lub katalogów wskazanych drugim i kolejnymi argumentami wywołania.

Uwaga: z oczywistych powodów zmiana praw pliku nie zawsze będzie możliwa – na przykład nie można w ten sposób przydzielić sobie uprawnień do cudzego pliku.

Specyfikacja uprawnień, jakie plik ma nabyć po wykonaniu polecenia **`chmod`**,

możliwa jest na kilka sposobów, używanych zależnie od okoliczności. Pierwszy z nich posługuje się czytelnym (na pewno czytelniejszym od zapisu ósemkowego) formalizmem pozwalającym określić nadawane bądź odbierane prawa, a który można zapisać w skrócie jako

kto-jak-co

Kto wskazuje jakiej kategorii użytkowników dotyczy operacja:

- **u** – właściciel
- **g** – grupa
- **o** – pozostali
- **a** – wszyscy

Litery te mogą być łączone w dowolne zestawy, choć nie wszystkie możliwe zestawy mają sens praktyczny. Możliwe jest również użycie pustego zestawu, w takim przypadku uznaje się, że jest on tożsamy z **a**.

Jak opisuje operację, która ma zostać wykonana:

- **+** – dodanie uprawnień do już istniejących
- **–** – odjęcie uprawnień od już istniejących
- **=** – przypisanie uprawnień bez względu na to, jakie istniały wcześniej

Co specyfikuje uprawnienia, które są brane pod uwagę w czasie wykonywania operacji i używa się tutaj niepustego złożenia znaków **r**, **w** i **x**. Zestaw *kto-jak-co* może wystąpić więcej niż raz, ale w takim przypadku kolejne zestawy muszą być rozdzielone przecinkami, bez wtrąconych białych znaków. Pojawienie się w tym miejscu białego znaku będzie oznaczać, że kolejny ciąg niebiałych znaków jest już nazwą pliku, co z pewnością nie będzie zgodne z intencjami wydającego polecenie.

Oto przykładowe zlecenia z wykorzystaniem powyższej składni polecenia **`chmod`**:

- **`chmod u+w plik.txt`**
  dodaje prawo zapisu dla właściciela do pliku *plik.txt*
- **`chmod go-x skrypt`**
  usuwa prawo wykonywania pliku *skrypt* przez użytkowników z grupy właścicielskiej i innych
- **`chmod a=r fb`**
  ustawia prawa dostępu do pliku **`fb`** na „*tylko do odczytu*” dla wszystkich użytkowników
- **`chmod u+x,g-x wirus1 wirus2`**
  nadaje prawo wykonywania plików *wirus1* i *wirus2* przez właściciela i odbiera to prawo grupie

Dygresja: zwróć uwagę, że prawo wykonania jest tak naprawdę czymś więcej, niż tylko zezwoleniem bądź jego brakiem. Można powiedzieć, że nadanie tego

prawa de facto czyni plik wykonywalnym, jako że w odróżnieniu od systemów klasy Windows informacja o tym, czy plik jest wykonywalny ***nie pochodzi z jego nazwy****. Oznacza to także, że każdy plik (w tym bitmapa) może być z punktu widzenia systemu rozpoznawany jako wykonywalny. Sens takiego działania jest raczej dyskusyjny, ale jest to możliwe.*

Polecenie **`chmod`** umożliwia także określanie praw dostępu wprost w postaci ósemkowej, jednak w tym przypadku nie ma możliwości modyfikowania pojedynczych uprawnień i zawsze należy podać komplet praw pliku/katalogu. Na przykład:

- **`chmod 777 file*`**
  nadaje do plików, których nazwy zaczynają się od liter *file*, wszystkie możliwe uprawnienia wszystkim kategoriom użytkowników
- **`chmod 742 you`**
  ustawia prawa odczytu, zapisu i wykonywania właścicielowi, prawo odczytu użytkownikom z tej samej grupy oraz prawo zapisu innym użytkownikom do pliku *you*

Poniżej prezentujemy zapis kilku operacji zmiany praw pliku wraz uwidocznieniem uzyskanych efektów. Analizę tej sekwencji poleceń pozostawiamy czytelnikowi.

wykonane polecenie	stan uprawnień po wykonaniu polecenia
`touch plik`	**`644 -rw-r-r-`**
`chmod +x plik`	**`755 -rwxr-xr-x`**
`chmod g+w,o-x plik`	**`774 -rwxrwxr-`**
`chmod 755 plik`	**`755 -rwxr-xr-x`**
`chmod 640 plik`	**`640 -rw-r---`**

2.1.2. Polecenie `chown`

`chown` ***[przełączniki...] właściciel[:grupa] plik...***

Polecenie **`chown`** umożliwia zmianę właściciela lub grupy pliku.

Uwaga: nie każda zmiana właściciela jest dopuszczalna. Z całą pewnością nie możesz w ten sposób przywłaszczyć sobie pliku należącego do użytkownika root. Co więcej, nie możesz również w ten sposób uczynić użytkownika root właścicielem swojego pliku. Z powyższych powodów pełnia mocy tego polecenia ujawnia się dopiero wtedy, gdy jest używane przez administratora, któremu zezwala się na dowolne przewłaszczanie plików.

2.1.3. Polecenie `chgrp`

`chgrp` ***[przełączniki...] grupa plik...***

Zmienia grupę, do której należy wskazany plik lub katalog. Obowiązują te same

zastrzeżenia, jak przy poleceniu **chown**.

2.2. Użytkownicy i grupy – reprezentacja w systemie

W systemach klasy UNIX informacja o użytkownikach jest tradycyjnie przechowywana w plikach tekstowych o znormalizowanej strukturze, nazwie i lokalizacji. Jeżeli konkretna instalacja tego wymaga, możliwe jest również stosowanie zamiast tego innych rozwiązań, na przykład w postaci systemów katalogowych lub baz danych, jednak nie są to rozwiązania typowe i raczej niespotykane w instalacjach desktopowych.

Pliki te to:

- **/etc/passwd**
- **/etc/shadow**
- **/etc/groups**

Pliki te mają charakterystyczną dla systemów uniksowych strukturę wierszową, gdzie każdy z wierszy jest podzielony na pola przy pomocy separatora, którym jest znak **:** (dwukropek).

2.2.1. Plik /etc/passwd

Plik **/etc/passwd** przeznaczony do przechowywania informacji o:

- nazwie każdego, znanego systemowi, użytkownika (przy czym warto zaznaczyć, że istnienie pewnego konta użytkownika nie oznacza, że użytkownik ten jest człowiekiem – konta zakłada się również dla wybranych usług systemu, jeśli ich funkcjonowanie z uprawnieniami użytkownika *root* mogłoby zagrażać integralności systemu
- identyfikatorze użytkownika (UID, od ang. *user ID*)
- identyfikatorze grupy macierzystej tego użytkownika (GID, od ang. *group ID*)
- położeniu katalogu domowego użytkownika
- informacji osobistych (numer telefonu, numer pokoju, etc)
- nazwy programu powłoki, który ma zostać uruchomiony po zalogowaniu użytkownika (podanie w tym miejscu nazwy programu **/sbin/nologin** jest najprostszą metodą karnego zablokowania możliwości faktycznego zalogowania się, ponieważ **nologin** kończy pracę natychmiast z kodem zakończenia sugerującym wystąpienie błędu; tego samego mechanizmu używa się do uniemożliwienia zalogowania się na konto przypisane do usług systemu

Historycznie plik ten używany był również do przechowywania informacji o hasłach użytkowników, co jest o tyle intrygujące, że domyślne prawa tego pliku to **644** (właścicielem jest użytkownik *root*). Oznacza to, że zawarte w pliku informacje są dostępne dla wszystkich (*sic!*). Oczywiście, w pliku tym przechowywano nie same

hasła, a tak zwane *hasze* haseł (aby uczynić historię krótszą, powiedzmy tylko, że są to wyniki przekształcenia tekstu w tekst przy pomocy pewnej nieróżnowartościowej i nieodwracalnej funkcji; dodajmy też, że funkcja taka jest znana publicznie). W momencie logowania system obliczał hasz z wprowadzonego hasła i porównywał go z haszem pamiętanym w **/etc/passwd**. Niestety, bardzo szybko okazało się, że wiara w to, że odtworzenie oryginalnego hasła z hasza nie jest możliwe, jest co najmniej naiwna. Praktyka pokazała, że oryginalne hasło wcale nie jest potrzebne – wystarczy odnalezienie tak zwanej *kolizji*, tzn. ciągu znaków, który daje w wyniku taki sam hasz, a to otwiera na oścież podwoje systemu.

Narzucający się niemal natychmiast pomysł, aby po prostu zmienić prawa pliku **/etc/passwd** na przykład na **600** jest spóźniony – zbyt wiele narzędzi i usług zakłada, że plik ten jest publicznie odczytywalny. W tej sytuacji zdecydowano się na inne rozwiązanie: w pole przeznaczone na hasz hasła wpisuje się znak 'x' (który nie ma prawa wystąpić w pojedynkę w prawdziwym haszu hasła), a same hasze przechowuje się w pliku

/etc/shadow

który dla odmiany ma prawa **640** i którego nie da się już odczytać bez posiadania odpowiednio wysokich uprawnień.

Poniżej przytaczamy przykładowy obraz zapisu w pliku **/etc/passwd**, dotyczący hipotetycznego użytkownika o nazwie **user**.

```
user:x:1001:1002:Waldemar Kiepski:/home/user:/bin/bash
```

Powyższy zapis odczytujemy następująco:

- w systemie istnieje konto użytkownika o loginie **user**,
- hasz hasła tego użytkownika przechowywany jest jest w pliku **/etc/shadow**,
- UID tego użytkownika to **1001**
- GID grupy macierzystej tego użytkownika to **1002**
- użytkownik ów zwie się naprawdę **Waldemar Kiepski**
- jego katalog domowy to **/home/user**
- w momencie logowania tego użytkownika należy dla niego uruchomić powłokę **/bin/bash**

Z kolei wpis do pliku **/etc/shadow** odnoszący się do tego samego użytkownika może wyglądać jak poniżej (dla celów publikacyjnych skrócono znacząco zapis hasza hasła):

```
user:$6$GYVr5axz$BlCIPnhv3jkwHSRWuJLICmrHALQP/:18762:0:99999:7:::
```

Powyższy wpis interpretuje się następująco:

- wpis dotyczy użytkownika **user**,
- hasz hasła tego użytkownika to **6GYVr5axz$BlCIPnhv3jkwHSRWuJLICmrHALQP/**
- ostatnia zmiana hasła tego użytkownika dokonała się w dniu odległym o **18762** dób od 1 stycznia 1970 roku (czyli 15 maja 2021 roku)
- użytkownikowi wolno będzie zmienić hasło po upływie **0** dni (czyli może je zmienić kiedy chce)
- hasło użytkownika wygaśnie po upływie **99999** dni (czyli praktycznie nigdy)
- użytkownik zostanie uprzedzony o konieczności zmiany hasła na **7** dni przed jego wygaśnięciem
- pozostałe parametry dotyczące czasu wygaśnięcia konta nie zostały podane, a więc konto będzie aktywne aż do momentu jego usunięcia

2.2.2. Plik `/etc/group`

Plik ten przeznaczony jest do przechowywania informacji o grupach użytkowników znanych w systemie. Dla każdej grupy pamięta się:

- jej nazwę
- jej identyfikator (GID, od ang. *group ID*)
- listę nazw użytkowników będących członkami grupy

Standardowe prawa tego pliku to **644**.

Nie rozwijając tego wątku dalej dodajmy, że w ogólnym przypadku grupy mogą mieć własnych administratorów oraz własne hasła. Ze względu na niewielkie rozpowszechnienie wykorzystania tego mechanizmu nie będzie on dokładniej omawiany. Poprzestaniemy na wzmiance, że problemy z przechowywaniem haszy haseł były tu identyczne jak w przypadku pliku */etc/passwd* i identycznie też je rozwiązano, przenosząc wrażliwe dane do pliku **/etc/gshadow**.

Poniżej przytaczamy przykładowy obraz zapisu w pliku **/etc/group**, dotyczący grupy o nazwie **power**, zwyczajowo służącej do nadania określonym użytkownikom prawa do posługiwania się narzędziami do zarządzania zasilaniem.

```
power:x:87:user,mickey,ola
```

Powyższy zapis odczytujemy następująco:

– w systemie istnieje grupa o nazwie **power**,
– hasz hasła tej grupy jest przechowywany w pliku **/etc/gshadow**,
– GID tej grupy to **87**
– do grupy tej należą użytkownicy **user**, **mickey** i **ola**.

2.2.3. Zmiana własnego hasła

Zmianę własnego hasła wykonuje się z użyciem polecenia **passwd**. Zmianę hasła użytkownika innego niż uruchamiający polecenie może wykonać tylko użytkownik *root*.

Polecenie uruchamia się bez argumentów, sama zmiana hasła przebiega konwersacyjnie, a przebieg dialogu silnie zależy od stopnia zabezpieczeń wprowadzonych w konkretnej instalacji. Jeden z możliwych scenariuszy zaprezentowano poniżej (znakami XXX zamaskowano wprowadzane hasła – normalnie są one niewidoczne):

```
user@host $ passwd
Changing password for user.
Current password: XXXXXXXXXXXXXXX

You can now choose the new password or passphrase.

A valid password should be a mix of upper and lower case letters, digits, and
other characters.  You can use a password containing at least 7 characters
from all of these classes, or a password containing at least 8 characters
from just 3 of these 4 classes.
An upper case letter that begins the password and a digit that ends it do not
count towards the number of character classes used.

A passphrase should be of at least 3 words, 11 to 72 characters long, and
contain enough different characters.

Alternatively, if no one else can see your terminal now, you can pick this as
your password: "lofty6Trough&Bogus".

Enter new password: XXXXXXXXXXXX
Re-type new password: XXXXXXXXXXXX
passwd: password updated successfully
```

2.3. i-węzły i dowiązania

W systemach klasy UNIX metadane plików przechowywane są w strukturach, które nazywa się i-węzłami (ang. *i-node*, od *index-node*). Każdy taki i-węzeł przechowuje

m. in. następujące metadane:

- prawa dostępu
- czas ostatniej zmiany zawartości lub metadanych pliku (ang. *ctime – last change time*) – sytuacja taka może zajść na przykład na skutek zmiany nazwy albo praw pliku, ale także po zapisie do pliku
- czas ostatniej modyfikacji zawartości pliku (ang. *mtime – last modification time*) – na przykład na skutek dopisania nowych danych na koniec pliku
- czas ostatniego dostępu do pliku (ang. *atime – last access time*) – za dostęp taki uznaje się również odczyt z pliku
- rozmiar pliku (w bajtach)
- licznik dowiązań (to zagadnienie omówimy wkrótce)
- inne atrybuty, w tym zależne od konkretnej wersji systemu operacyjnego

Zauważ, ze pomiędzy wymienionymi powyżej trzema czasami zachodzi następująca zależność:

- *atime* może się zmienić jako jedyny spośród tej trójki (na przykład na skutek odczytu z pliku)
- zmiana *ctime* automatycznie pociąga za sobą zmianę *atime* (na przykład po zmianie nazwy pliku)
- zmiana *mtime* wywołuje jednoczesną zmianę *atime* i *mtime* (na przykład po zapisie do pliku)

Warto dodać, że każdy z powyższych czasów reprezentowany jest w systemie plików jako dana całkowita bez znaku, obrazująca liczbę sekund od północy czasu uniwersalnego 1 stycznia 1970 roku, co umownie uznaje się za początek tak zwanej *epoki Uniksa* (ang. *Unix Epoch*).

Należy jednak zwrócić uwagę na fakt, że konsekwentne i pełne aktualizowanie *atime* każdego pliku w systemie może być niezwykle kosztowne, jako że wymaga ogromnej ilości zapisów do wszystkich zaangażowanych i-węzłów. Z tego też powodu konkretna instalacja może minimalizować aktualizację *atime* tylko do wybranych sytuacji, na przykład do operacji zamknięcia pliku otwartego wcześniej do czytania.

Licznik dowiązań określa **ile razy** dany plik jest dostępny w systemie plików (być może w różnych katalogach i pod różnymi nazwami). Licznik ten umożliwia realizację tak zwanych *dowiązań do plików* (ang. *links*), które to dowiązania są dodatkowymi nazwami (aliasami) nadanymi plikowi, umożliwiając dostęp do oryginału na przykład z poziomu różnych katalogów albo, na przykład w celu wybrania jednej z wielu dostępnych wersji plików.

Istnieją dwa rodzaje dowiązań:

- dowiązania **twarde** (ang. *hard links*), realizowane fizycznie na poziomie organizacji i-węzłów wewnątrz systemu plików w sposób niewidoczny dla

użytkownika
- dowiązania **miękkie**, zwane również **symbolicznymi** (ang. *soft* lub *symbolic links*), realizowane poprzez pliki specjalnego przeznaczenia, reprezentujące dowiązanie w sposób logiczny

Podstawowa różnica w działaniu obu rodzajów dowiązań polega na tym, że dowiązania twarde mogą funkcjonować tylko w obrębie tego samego systemu plików, który dla potrzeb niniejszych rozważań możemy utożsamić z partycją. Dowiązania symboliczne mogą natomiast przekraczać granice partycji i odnosić się także do plików umieszczonych na innych urządzeniach fizycznych.

Informacja o dowiązaniu dostępna jest dzięki omówionemu już poleceniu **`ls -l`**. Wszystkie dowiązania można przetwarzać dokładnie tak samo jak odpowiadające im pliki i katalogi, w szczególności mogą także być usunięte poleceniem rm (zauważ, że w ten sam sposób, a nie poleceniem **`rmdir`** usuwa się również dowiązanie do katalogu).

Fakt, że pewien plik jest dowiązaniem symbolicznym najłatwiej zarejestrować analizując wyjście z polecenie **`ls`** z przełącznikiem **`-l`**, na przykład:

```
$ ln -s plik symboliczne
$ ls -l
lrwxrwxrwx 1 user   user   4 01-01 12:00 symboliczne -> plik
-rw-r--r-- 1 user   user   0 01-01 11:00 plik
```

Zauważ, że:
- dowiązanie symboliczne o nazwie **`symboliczne`** ma literę **`l`** (od ang. *link*) na pierwszym znaku maski praw, i to niezależnie od tego, czy prowadzi do pliku, czy do katalogu,
- dowiązanie takie ma komplet możliwych praw, co jednak nie ma znaczenia w przypadku korzystania z niego, bowiem dowiązanie symboliczne ma faktycznie dokładnie takie same prawa, jak plik/katalog do którego prowadzi,
- za nazwą dowiązania symbolicznego, a po strzałce, prezentowana jest nazwa pliku/katalogu, do którego dowiązanie prowadzi,
- rozmiar dowiązania symbolicznego jest równy długości nazwy pliku/katalogu, do którego dowiązanie prowadzi (z ewentualną poprzedzającą ścieżką) i nie ma nic wspólnego z faktycznym rozmiarem dowiązanego obiektu,
- warte zauważenia jest, że dowiązanie symboliczne może mieć inny znacznik czasu niż plik, do którego prowadzi.

Dla odmiany, dowiązanie twarde jest nieodróżnialne od pliku/katalogu, co potwierdza, że ten twór jest w istocie aliasem nazwy pliku reprezentowanym na

poziomie systemu plików.

```
$ ln plik twarde
$ ls -l
-rw-r--r-- 1 user   user   0 01-01 11:00 plik
-rw-r--r-- 1 user   user   0 01-01 11:00 twarde
```

Jak widać, dowiązanie twarde ma taki sam znacznik czasu co plik, do którego prowadzi.

- dowiązanie twarde o nazwie **twarde** nie ma żadnego specjalnego wyróżnika na pierwszym znaku maski praw (dodajmy, że dowiązanie takie może prowadzić wyłącznie do pliku),
- dowiązanie takie ma dokładnie takie sama prawa jak plik docelowy,
- rozmiar dowiązania twardego jest równy rozmiarowi pliku docelowego.

Tworzenie dowiązań jest możliwe dzięki poleceniu **ln** (ang. *link*), którego najczęściej stosowana postać prezentuje się jak poniżej:

ln *[przełącznik...] plik_faktyczny nazwa_dowiązania*

Argument *plik_faktyczny* musi wskazywać na istniejący plik (lub katalog w przypadku dowiązań symbolicznych), do którego tworzone jest dowiązanie, a drugim argumentem jest nazwa, pod którą zostanie utworzone nowe dowiązanie. Utworzenie dowiązania symbolicznego wymaga zastosowania przełącznika **-s**.

Oto kilka przykładów wywołania zlecenia utworzenia dowiązań:

ln ./abc/plik.txt plik1.txt
tworzy w katalogu bieżącym dowiązanie **twarde** do pliku *plik.txt* w katalogu *./abc* pod nazwą *plik1.txt*

ln -s abc/plik.txt /plik1.txt
tworzy w katalogu domowym użytkownika dowiązanie **symboliczne** do pliku *plik.txt* w katalogu *abc* pod nazwą *plik1.txt*

Inne, rzadziej stosowane, formy polecenia **ln** to:

ln *[przełącznik...]* plik_faktyczny
tworzy w katalogu bieżącym dowiązanie nazwie takiej, jaką nosi *plik_faktyczny*; z oczywistych przyczyn *plik_faktyczny* musi znajdować się poza katalogiem bieżącym

ln *[przełącznik...] plik_faktyczny... katalog*
we wskazanym *katalogu* dla każdego z *plików_faktycznych* tworzy dowiązanie o takiej samej nazwie jak *plik_faktyczny*

ln *[przełącznik...]* -t *katalog plik_faktyczny...*
Działanie jak w poprzednim przypadku.

Dygresja. Ciekawą cechą dowiązań jest to, że jeśli pewien plik używany jest poprzez któreś ze swoich dowiązań, to traktowany jest tak, jakby nosił nazwę dowiązania, a nie nazwę pliku faktycznego. W szczególności, w programach napisanych w języku C parametr ***argv[0]*** *będzie zawierał nazwę użytą aby uruchomić dany program, a nie nazwę pliku, w którym znajduje się kod wykonywalny. Oznacza to, że ten sam kod może wypełniać różne funkcje w zależności od nazwy dowiązania, jakie spowodowało jego uruchomienie. Ta cecha dowiązań wykorzystywana jest na przykład przez program o charakterystycznej nazwie* ***busybox****, dołączany standardowo do praktycznie wszystkich dystrybucji Linuksa, który potrafi zachowywać się jak polecenia* ***cp****,* ***mv****,* ***rm*** *i tym podobnych, o ile tylko zostanie uruchomiony taką właśnie nazwą. Rozwiązanie takie pozwala oszczędzić miejsce na nośniku w przypadku zastosowania go w rozwiązaniach typu live, dystrybuowanych na mediach o ograniczonej pojemności (na przykład w plikach ISO przeznaczonych do bezpośredniego zabootowania systemu) bądź też w systemach wbudowanych, gdzie każdy bajt nośnika jest na wagę złota.*

2.4. Polecenie **date**

Polecenie **date** pozwala każdemu użytkownikowi odczytać bieżący czas systemowy, a użytkownikowi o odpowiednio wysokich uprawnieniach umożliwia również tenże czas ustawić. Z tego tez powodu omówimy tu wyłącznie te formy polecenia **date**, które mogą być użyte przez wszystkich:

date ***[przełączniki...] [+FORMAT]***

- użyte bez argumentów wyświetli bieżący czas systemowy w formacie zgodnym z ustawioną „narodowością” systemu i z użyciem nazw miesięcy oraz dni tygodnia w stosownym języku naturalnym
- przełącznik **-R** spowoduje wyświetlenie czasu w formacie RFC 2822 (używanym na przykład w poczcie elektronicznej)
- przełącznik **-u** spowoduje wyświetlenie czasu przekształconego do czasu uniwersalnego (UTC)
- przełącznik **-r** z parametrem będącym nazwą pliku podaje czas ostatniej modyfikacji tego pliku
- parametr **FORMAT**, jeśli zostanie użyty, jest ciągiem znaków formatujących, podobnych w swojej postaci do formaterów używanych przez funkcję **printf()** z języka C; poniżej prezentujemy kilka najważniejszych specyfikatorów używanych przez polecenie date:
  - **%a** lokalny skrót nazwy dnia tygodnia (na przykład **nie**)
  - **%A** lokalna pełna nazwa dnia tygodnia (na przykład **niedziela**)
  - **%b** lokalny skrót nazwy miesiąca (na przykład **sty**)
  - **%B** lokalna pełna nazwa miesiąca (na przykład **styczeń**)

%c lokalna data i czas (na przykład **czw mar 3 23:05:25 2005**)
%C stulecie, jak **%Y**, tylko z obciętymi ostatnimi dwiema cyframi
%d dzień miesiąca (na przykład **01**)
%D data, to samo co **%m/%d/%y**
%e dzień miesiąca uzupełniony spacjami, jak **%_d**
%F pełna data; to samo co **%Y-%m-%d**
%g ostatnie dwie cyfry roku numeru tygodnia ISO (patrz **%G**)
%H godzina (**00..23**)
%I godzina (**01..12**)
%j dzień roku (**001..366**)
%k godzina, ewentualnie uzupełniona spacją z lewej (**0..23**); to samo co **%_H**
%l godzina, uzupełniona spacją (**1..12**); to samo co **%_I**
%m miesiąc (**01..12**)
%M minuta (**00..59**)
%n znak nowego wiersza
%N nanosekundy (**000000000..999999999**)
%p lokalny odpowiednik **AM** lub **PM**; w wielu językach (w tym w polskim) pusty
%P jak **%p**, lecz małymi literami
%r czas w formacie 12-godzinnym (na przykład **11:11:04 PM**)
%R godzina i minuty w formacie 24-godzinnym; to samo co **%H:%M**
%s liczba sekund od północy 1 stycznia 1970 UTC
%S sekunda (**00..60**)
%t tabulator
%T czas; to samo co **%H:%M:%S**
%u dzień tygodnia (**1..7**); 1 to poniedziałek
%U numer tygodnia w roku, przy założeniu, że tydzień zaczyna się od niedzieli (**00..53**)
%V numer tygodnia według normy ISO 8601:2004, przy założeniu, że tydzień zaczyna się od poniedziałku (**01..53**); ze względu na powszechność stosowania tej konwencji (jest ona honorowana także w Polsce), przytoczymy ją w pełnym brzmieniu:

  – tydzień trwa 7 dni
  – pierwszym dniem tygodnia jest poniedziałek
  – pierwszym tygodniem danego roku jest tydzień zawierający pierwszy czwartek tegoż roku

%w dzień tygodnia (**0..6**); 0 oznacza niedzielę
%W numer tygodnia w roku, przy założeniu, że tydzień tydzień zaczyna się

od poniedziałku (**00..53**)
%x lokalna reprezentacja daty (na przykład **19.03.2012**)
%X lokalna reprezentacja czasu (na przykład **23:13:48**)
%y dwie ostatnie cyfry roku (**00..99**)
%Y rok

Domyślnie, **date** dopełnia pola numeryczne zerami. Następujące opcjonalne modyfikatory mogą być umieszczone po **%**:

- (łącznik) bez wypełniania pola
_ (podkreślenie) wypełnianie spacjami
0 (zero) wypełnianie zerami

Ponadto, można również wpływać na rejestr liter używanych do prezentowania tekstu:

^ używaj wielkich liter
zamieniaj wielkość liter

2.5. Inne przydatne polecenia

2.5.1. Polecenie stat

Polecenie **stat** służy do prezentacji kompletu metadanych pliku/katalogu i w najprostszym wariancie jest uruchamiane w sposób następujący:

```
stat plik...
```

Poniżej prezentujemy postać wyjścia tego polecenia w przypadku, gdy jego argumentem jest zwykły plik:

```
$ stat plik
Plik: plik
  rozmiar: 905688     bloków: 1776       bloki I/O: 4096   plik zwykły
Urządzenie: 10308h/66312d inody: 8085747     dowiązań: 1
Dostęp: (0755/-rwxr-xr-x)  Uid: ( 1000/  user)   Gid: ( 1000/  user)
Dostęp:      2022-09-21 14:18:55.676384812 +0200
Modyfikacja: 2022-09-21 14:18:55.677384811 +0200
Zmiana:      2022-09-21 14:18:55.677384811 +0200
Utworzenie:  2022-09-21 14:18:55.676384812 +0200
```

Jak widać, zestawienie prezentuje następujące metadane pliku:
– nazwę pliku (**plik**)
– rozmiar pliku w bajtach (**905688**)

- liczbę bloków alokacji przydzielonych do pliku (**1776**)
- rozmiar bloku alokacji w bajtach (**4096**)
- identyfikator urządzenia, na którym rezyduje plik, y szesnastkowo (**10308h**) i dziesiętnie (**66312d**); nie wnikając głębiej w interpretację znaczenia tej danej nadmieńmy, że jeśli dwa pliki mają ten sam numer urządzenia, to rezydują na tej samej partycji
- numer pierwszego i-węzła pliku (**8085747**)
- liczbę dowiązań twardych prowadzących do pliku; paradoksalnie, wartość **1** oznacza, że plik nie ma dodatkowych dowiązań
- prawa pliku w postaci ósemkowej (**0755**) i znakowej (**-rwxr-xr-x**)
- UID (**1000**) i nazwę (**user**) właściciela pliku
- GID (**1000**) i nazwę (**user**) grupy właścicielskiej pliku
- kolejno, cztery znaczniki czasu informujące o ostatnim dostępie do pliku, ostatniej modyfikacji zawartości, zmianie atrybutów i utworzeniu

Postać informacji prezentowanej przez **stat** może zostać sformatowana w inny sposób, jeśli użytkownik wykorzysta przełącznik **-c** lub jego długą formę **--format**, a następnie poda ciąg znaków formatujących wybranych z poniższego zestawu (zaprezentowano tylko te warianty, które mają odbicie w standardowej postaci wyjścia polecenia **stat**):

%a prawa pliku w postaci ósemkowej
%A prawa pliku w postaci znakowej
%b liczba przydzielonych bloków
%B rozmiar bloku alokacji w bajtach
%d numer urządzenia dziesiętnie
%D numer urządzenia szesnastkowo
%F rodzaj pliku w postaci tekstowej (na przykład **plik zwykły**)
%g GID grupy właścicielskiej
%G nazwa grupy właścicielskiej
%h liczba twardych dowiązań prowadzących do pliku
%i numer pierwszego i-węzał pliku
%n nazwa pliku
%s rozmiar pliku w bajtach
%u UID właściciela
%U nazwa właściciela
%w czas utworzenia pliku (postać czytelna dla ludzi)
%x czas ostatniego dostępu do pliku (postać czytelna dla ludzi)
%y czas ostatniej modyfikacji zawartości pliku (postać czytelna dla ludzi)
%z czas ostatniej modyfikacji atrybutów pliku (postać czytelna dla ludzi)

Poniżej prezentujemy wzór użycia sekwencji formatującej zastosowanej do

tego samego pliku, co w poprzednim przykładzie:

```
$ stat -c '%n: %s bajtów, %b bloków' plik
plik: 905688 bajtów, 1776 bloków
```

2.5.2. Polecenie `sync`

Polecenie **`sync`** użyte bez argumentów wymusza opróżnienie wszystkich buforów zapisu i przeniesienie ich zawartości do systemów plików oraz zaktualizowanie informacji pamiętanych w i-węzłach. Używane, gdy użytkownik chce mieć pewność, że wszystkie zainicjowane wcześniej operacje zapisu (na przykład szczególnie kierowane na powolne urządzenia wymienne takie jak pendrajwy albo karty SD) zostały faktycznie zakończone.

2.5.3. Polecenie `df`

`df` ***`[przełącznik...] [plik...]`***

Powoduje wyświetlenie informacji o dostępnej wolnej przestrzeni w systemie plików. Jeśli nie podano argumentu/argumentów *plik...*, prezentowana jest informacja o wszystkich systemach plików znanych w danym momencie systemowi operacyjnemu. Jeśli argument/argumenty podano, prezentowana jest tylko informacja o systemie/systemach plików, na których rezydują pliki o podanych nazwach.

Na sposób prezentowania tej informacji mają wpływ następujące przełączniki:

`--total`
wyświetla podsumowanie całości

`-h`
wyświetla rozmiary w formacie czytelnym dla ludzi

`-H`
podobnie, ale używa potęg 1000 zamiast 1024

`--no-sync`
nie wywołuje **`sync`** przed pozyskaniem informacji (zachowanie domyślne), co powoduje, że prezentowane informacje mogą odbiegać od stanu faktycznego, ale ich uzyskanie przebiega natychmiast (w pewnych przypadkach i przy korzystaniu z powolnych urządzeń pełne wykonanie synchronizacji może trwać nawet kilkadziesiąt sekund)

`--sync`
wywołuje **`sync`** przed pozyskaniem informacji o wolnym nośniku

2.5.4. Polecenie `du`

`du` ***`[przełącznik...] [plik...]`***

W pewnym sensie jest to dopełnienie polecenia **`df`**, ponieważ powoduje wyświetlenie informacji o przestrzeni **zajętej** przez pliki lub katalogi. W obliczeniach uwzględniany jest nie tylko rozmiar pliku, ale również fakt, że plik może zajmować na nośniku więcej bądź mniej miejsca niż, wymaga tego jego zawartość. Jest to konsekwencja faktu, że nośnik dla pliku przydzielany jest kwantami o stałym rozmiarze i ostatni z przydzielonych kwantów nie musi być zajęty w całości przez dane. Inną okolicznością, która musi być tu wzięta pod uwagę, jest fakty istnienia tak zwanych *plików rzadkich* (ang. *sparse files*), dopuszczalnych w wielu współczesnych systemach plików. Plik rzadki to plik, wypełniony danymi tylko częściowo, przy czym puste (niewypełnione danymi) partie pliku zastępuje się krótkimi metadanymi, dzięki czemu plik taki zajmuje znacznie mniej nośnika.

Wynika z tego, że w pełni możliwe (choć nierównie prawdopodobne) są wszystkie poniższe trzy sytuacje:

- zajętość nośnika jest równa rozmiarowi pliku
- zajętość nośnika jest większa od rozmiaru pliku (ostatni kwant przydziału nie został zapełniony)
- zajętość nośnika jest mniejsza od rozmiaru w pliku (plik jest rzadki).

Warto zwrócić tu uwagę na fakt, że z punktu widzenia polecenia **`du`** rozmiar pliku wykazywany na przykład przez polecenia **`ls`** albo **`stat`**, jest tak zwanym *rozmiarem pozornym* (ang. *apparent size*).

Na sposób, w jaki **`du`** prezentuje informacje, mają wpływ następujące przełączniki:

`-a`
wypisuje podsumowania dla wszystkich plików, nie tylko katalogów

`--apparent-size`
wyświetla rozmiar pozorny zamiast faktycznego użycia dysku

`Bs`
przelicza rozmiary według jednostki *s*; dopuszczalne postaci *s* to:

- liczba całkowita
- liczba całkowita z przyrostkiem **`K`** (*kibi*), **`M`** (*mebi*), **`G`** (*gibi*), **`T`** (*tebi*), **`P`** (*pebi*), **`E`** (*exbi*), **`Z`** (*zebi*), **`Y`** (*yobi*) czyli z użyciem jednostek wynikających z potęg liczby 1024 albo **`KB`**, **`MB`**, itd., czyli przy pomocy jednostek branych z potęgi liczby 1000
- sam przyrostek

`-b`
równoważne **`--apparent-size -B1`**

`-c`

wyświetla podsumowanie całości

-D

rozwija tylko dowiązania symboliczne podane jako argumenty

-h

wyświetla rozmiary w formacie czytelnym dla ludzi (na przykład **1 KiB**, **234 MiB**, **2 GiB**)

--si

jak **-h**, lecz używa potęg 1000 zamiast 1024

-k

jak **-B1K**

-m

jak **-B1M**

-L

rozwija wszystkie dowiązania symboliczne

-P

nie podąża za żadnymi dowiązaniami symbolicznymi (domyślnie)

-S

nie uwzględnia rozmiarów podkatalogów

-s

wyświetla tylko podsumowanie dla każdego podanego argumentu

--exclude=*nazwa*

pomija pliki pasujące do wzorca podanego w *nazwa*

Poniżej zaprezentujemy wszystkie trzy możliwości wzajemnej relacji rozmiaru faktycznego i pozornego pewnego pliku. Zakładamy, że poniższy eksperyment przeprowadzany jest na systemie plików, w którym rozmiar alokacji wynosi 4096 bajtów (4 KiB).

W pierwszym kroku utworzymy plik o nazwie **plik**, którego faktyczna zawartość będzie miała rozmiar 16 bajtów. Możemy to uczynić wykorzystując polecenie **echo** i mechanizm przekierowania, omówione dokładnie w kolejnych rozdziałach tego dokumentu.

```
$ echo -n '0123456789ABCDEF' > plik
$ ls -l plik
-rwxr-xr-x 1 user user 16 01-01 12:00 plik
$ du -h plik
4,0K plik
$ u --apparent-size -h plik
16 plik
```

Jak widać, mamy tu do czynienia z sytuacją, w której rozmiar danych w pliku (rozmiar pozorny) równy 16 bajtów jest mniejszy od zajętości nośnika (4 KiB).

W drugim kroku wypełnimy plik danymi o rozmiarze równym rozmiarowi bloku alokacji, wykorzystując do tego celu zaawansowane narzędzie o nazwie **dd**. Nie wnikając głębiej w możliwości tego niezwykle uniwersalnego programu powiedzmy, że użyjemy go, aby z pliku **/dev/zero** – który jest niewyczerpanym źródłem bajtów o wartości zero – odczytać dane o rozmiarze 4096 bajtów i zapisać je w pliku **plik**.

```
$ dd if=/dev/zero of=plik bs=1 count=4096
4096+0 przeczytanych rekordów
4096+0 zapisanych rekordów
skopiowane 4096 bajtów (4,1 kB, 4,0 KiB), 0,0061595 s, 665 kB/s
$ ls -l plik
-rwxr-xr-x 1 user user 4096 01-01 12:00 plik
$ du -h plik
4,0K plik
$ du --apparent-size -h plik
4,0K plik
```

W tym przypadku rozmiar danych w pliku (rozmiar pozorny) równy 4096 bajtów jest dokładnie równy zajętości nośnika (4 KiB).

Do wywołania trzeciej możliwości użyjemy polecenie **truncate**, które zdolne jest zmienić rozmiar danych w pliku bez zmiany liczby przydzielonych do niego bloków. Mimo zwodniczej nazwy, polecenie potrafi nie tylko plik obciąć, ale również rozszerzyć. Wykorzystamy je do ustalenia rozmiaru przechowywanych w pliku na 8192 bajty. Tym samym plik **plik** stanie się plikiem rzadkim.

```
$ truncate --size=8192 plik
$ ls -l plik
-rwxr-xr-x 1 user user 8192 01-01 12:00 plik
$ du -h plik
4,0K plik
$ du --apparent-size -h plik
8,0K plik
```

Obserwujemy tutaj paradoksalną na pierwszy rzut oka sytuację, w której rozmiar danych w pliku (rozmiar pozorny) równy 8192 bajtów jest dwukrotnie większy od zajętości nośnika (nadal równej 4 KiB).

2.5.5. Polecenie `file`

`file` ***`[przełącznik...] plik...`***

To pożyteczne narzędzie istnieje w systemach klasy UNIX głównie z powodów doktrynalnych. Otóż filozofia tego systemu nigdy nie przewidywała, aby w nazwie pliku miała się znajdować jakakolwiek informacja o tym, co plik zawiera (co od zarania dziejów było standardem w systemach klasy Windows). Rozwój Internetu zmienił to nastawienie i w chwili obecnej używanie identyfikujących plik przyrostków jest nie tylko na porządku dziennym, ale ponadto zostało ujęte w normach międzynarodowych, jednak zawartość pewnych plików nadal może być zagadką. Na przykład pliki, które oznaczone są jako **`x`** (wykonywalne) mogą mieć bardzo różną zawartość: mogą być programami skompilowanymi, nazywanymi w slangu *binarkami*, ale mogą być też zwykłymi plikami tekstowymi, zawierającymi skrypty w przeróżnych językach interpretowanych (Bash, awk, tcl, Python, Perl, itd.). Można oczywiście próbować oglądać takie pliki od środka i wydawać osąd na podstawie wzrokowej analizy ich zawartości, ale nie zawsze jest to wygodne. Ponadto, może się również zdarzyć, że na skutek nieszczęśliwego wypadku plik może stracić oryginalną nazwę i warto posiadać narzędzie, które samo przeanalizuje jego zawartość i postawi hipotezę odnośnie tego, czym ten plik jest w istocie.

Polecenie **`file`** wykonuje tę czynność na postawie analizy zawartości specjalnej bazy danych nazywanej *magic numbers*. Jest to zbiór prostych heurystyk pozwalających sprawdzić, czy dany plik ma (lub nie) pewne ewidentne cechy (na przykład zbiór użytych znaków, obecność charakterystycznych sygnatur i tym podobne). Same heurystyki zapisywane są w plikach tekstowych umieszczanych zwyczajowo w katalogu **`/usr/share/misc/magi`**c, a ich lektura dostarcza bardzo ciekawych informacji na temat tego, jak dalece wiarygodne (bądź niewiarygodne) są diagnozy stawiane przez to narzędzie.

W najprostszym przypadku wystarczy wskazanie nazwy interesującego nas pliku, a **`file`** odpowie stosowną informacją, na przykład:

```
$ file win/explorer.exe
win/explorer.exe: PE32+ executable (GUI) x86-64, for MS Windows
```

co w tym konkretnym przypadku oznacza, że diagnozowany plik jest według heurystyk narzędzia **file** 64-bitowym plikiem wykonywalnym pochodzącym z systemu operacyjnego MS Windows, a przeznaczonym do uruchomienia na platformie Intel x86.

2.5.6. Polecenie sleep

sleep ***liczba[przyrostek]***

Polecenie **sleep** wywołuje efekt bezczynności trwający przez czas określany argumentem. Przydatne bywa głównie w skryptach, ale może być użyteczne także w pracy konsolowej na przykład w celu wykonania kolejnego polecenia dopiero po upływie pewnego czasu. Dopuszczalne przyrostki to:

- **s** sekundy (domyślne)
- **m** minuty
- **h** godziny
- **d** dni

Warto pamiętać, że wykonanie polecenia sleep może zostać w dowolnym momencie przerwane przez kombinację klawiszy *Ctrl-C*.

2.5.7. Polecenie diff

diff ***[przełącznik...] plik1 plik2***

Polecenie **diff** służy do porównania zawartości wskazanych plików – w najprostszym przypadku porównuje zawartość dwóch plików, *plik1* i *plik2*. Jeśli *plik1* jest katalogiem, a *plik2* nie, **diff** porównuje ten plik z katalogu *plik1*, którego nazwa jest taka sama, jak *plik2* i odwrotnie. Jeśli *plik1* i *plik2* są katalogami, **diff** porównuje pliki o zgodnych nazwach, istniejące w obu katalogach, w kolejności alfabetycznej. To porównanie nie jest ponawiane w podkatalogach, chyba że podano opcję **-r**. Jeśli porównywane pliki nie różnią się, **diff** nie odzywa się ani słowem.

Informacje wyprowadzanie przez **diff** mogą być bezpośrednio użyte przez polecenie **patch**, co umożliwia na oszczędne dystrybuowanie przyrostowych poprawek zamiast kompletnych, obszernych plików źródłowych. Mechanizm ten wykorzystuje się na przykład przy publikowaniu nieznacznych poprawek jądra Linuksa. Zapoznanie się z możliwościami narzędzia **patch** pozostawia się czytelnikowi.

Należy podkreślić, że **diff** został stworzony do wyszukiwania różnic w plikach tekstowych, a postać dostarczanych przez niego informacji ułatwia, na przykład identyfikowanie miejsc, w których zmodyfikowano kod źródłowy. Z tego też powodu **diff** nie nadaje się do porównywania plików binarnych (na przykład plików graficznych). Do tego celu służy polecenie **cmp**.

Najczęściej używane przełączniki **diff** to:

-i ignorowanie zmian w wielkości liter, wielkie i małe litery są uznawane za równe

-w ignorowanie wszystkich odstępów przy porównywaniu plików

-b ignorowanie zmian w ilości odstępów

-B ignorowanie zmian, które jedynie dodają lub usuwają puste linie

-q poinformowanie jedynie o tym, czy pliki się różnią, bez podawania szczegółów na temat różnic

-r rekurencyjne porównanie wszystkich podkatalogów, jeśli porównywane są katalogi

-N jeśli podczas porównywania katalogów plik istnieje jedynie w jednym z katalogów, będzie traktowany tak, jakby był obecny w drugim katalogu, ale jako plik pusty

-P jeśli podczas porównywania katalogów plik istnieje jedynie w drugim katalogu, będzie traktowany tak, jakby był obecny w pierwszym katalogu, ale jako plik pusty

-s poinformowanie, jeśli oba pliki są identyczne (normalnie w takiej sytuacji **diff** nie odzywa się ani słowem)

-x *wz* podczas porównywania katalogów, ignorowanie plików i podkatalogów, których nazwy pasują do wzorca *wz*

Informacje wyprowadzane z polecenia **diff** prezentowane są w dość pokrętny sposób, dlatego postaramy się je omówić na prostych przykładach, licząc jednocześnie, że czytelnik będzie skłonny kontynuować te eksperymenty na bardziej złożonych przypadkach.

W dalszej części naszych rozważań założymy, że pierwszy z porównywanych plików będzie nazywany plikiem *lewym*, a drugi – plikiem *prawym*. Terminologia ta pozwala łatwo zidentyfikować znaczenie zwrotów strzałek, jakie pojawiają się na wyjściu polecenia **diff**.

W tym miejscu warto zauważyć, że informacje pojawiające się na wyjściu polecenia **diff** są nie tyle opisem wykrytych różnic, ile specyficzną receptą na to, co należy zrobić, aby oba porównywane pliki stały się identyczne. Sztywna i sformalizowana postać tej recepty pozwala na w miarę łatwe zaimplementowanie automatu, który takie ujednolicenie może przeprowadzać samodzielnie, z równą łatwością doprowadzając plik prawy do zgodności z plikiem lewym albo odwrotnie.

W ogólnym przypadku, jeśli wykryto różnice w porównywanych plikach, wyjście z polecenia **diff** zawiera jedno- lub więcej krotne wystąpienia bloku tekstu o następującej strukturze:

```
OPIS-RÓŻNICY
< LINIA-PLIKU-LEWEGO
< LINIA-PLIKU-LEWEGO...
---
> LINIA-PLIKU-PRAWEGO
> LINIA-PLIKU-PRAWEGO...
```

Element *OPIS-RÓŻNICY* może mieć jedną z trzech możliwych postaci:

`LaP` (gdzie '**`a`**' pochodzi od angielskiego słowa *addition* – dodanie) i ma następujące znaczenie:
*dodaj linie z przedziału opisanego przez **P** z pliku prawego za linią o numerze **L** w pliku lewym*; na przykład **`4a8,11`** oznacza, że w celu ujednolicenia obu plików należy za czwartą linią w pliku lewym dodać cztery linie (od ósmej do jedenastej) z pliku prawego,

`LcP` (gdzie '**`c`**' pochodzi od angielskiego słowa *change* – zmiana) i ma następujące znaczenie:
*wymień linie z przedziału opisanego przez **L** w pliku lewym na linie z przedziału **R** z pliku prawego*; na przykład **`2,3c6,8`** oznacza, że w celu ujednolicenia obu plików należy dwie linie w pliku lewym (drugą i trzecią) wymienić na trzy linie z pliku prawego (szóstą, siódmą i ósmą),

`LdP` (gdzie '**`d`**' pochodzi od angielskiego słowa *deletion* – usunięcie) i ma następujące znaczenie:
*usuń z pliku lewego wszystkie linie z przedziału opisanego przez **L** podczas gdy **R** opisuje miejsce w pliku prawym, gdzie linie te mogłyby się pojawić w celu uzyskania efektu identyczności*; na przykład **`2,3d4`** oznacza, że w celu ujednolicenia obu plików należy albo usunąć dwie linie w pliku lewym (drugą i trzecią), albo dodać je za linią czwartą w pliku prawym (oczywiście, mimo faktu, że obie alternatywne operacje doprowadzają do identyczności plików, efekt będzie diametralnie inny) ,

Poniżej prezentowane są wszystkie linie z obu plików, których dotyczy wcześniejszy opis, przy czym zwrot grotów strzałek wskazuje pochodzenie linii, a wiersz złożony z trzech łączników ułatwia odróżnienie tekstu pochodzącego z różnych plików.

Prześledźmy teraz trzy trywialne przykłady ilustrujące podane wyżej opisy.

2.5.8. Znacznik `a` (dodanie)

Załóżmy, że porównaniu podlegają pliki o nazwach *lewy* oraz *prawy* i o następującej zawartości:

lewy :

```
111
333
```

prawy :

```
111
222
333
```

Ponownie uruchamiamy polecenie **diff**:

```
$ diff lewy prawy
1a2
> 222
```

Uzyskany wynik ma następujące znaczenie:
- różnica dotyczy linii, która jest linią numer 2 w pliku prawym, a którą należy **dodać** za linią za linią numer 1 pliku lewego (*1a2*),
- poniżej zacytowano linię, którą występuje w pliku prawym, a jest nieobecna w pliku lewym,
- warto tu zauważyć, że znacznik **a** jest *de facto* złożeniem diagnoz podawanych przez znaczniki **d** i **c**, pozwala jednak na znaczne skrócenie rozmiaru prezentowanych wyników.

2.5.9. Znacznik c (wymiana)

Po modyfikacji zawartość obu plików prezentuje się teraz następująco:

lewy :

```
11
```

prawy :

```
111
```

W takim kontekście uruchamiamy teraz polecenie **diff**:

```
$ diff lewy prawy
1c1
< 11
---
> 111
```

Otrzymany wynik interpretujemy następująco:
- wykryto różnicę położoną w linii 1 pliku lewego, odpowiadającą linii 1 pliku prawego (*1c1*)
- poniżej zacytowano obie różniące się linie, wskazując z jakich plików pochodzą oraz rozdzielając je znacznikiem ---, co stanowi sugestię, że wymiana dowolnej z prezentowanych linii na drugą z nich sprawi, że pliki będą identyczne

2.5.10. Znacznik `d` (usunięcie)

Po kolejnej modyfikacji oba pliki mają teraz poniższą zawartość:

lewy :

```
111
222
```

prawy :

```
111
```

Sprawdzamy, jaki zmieniła się odpowiedź polecenia **`diff`**:

```
$ diff lewy prawy
2d1
< 222
```

Interpretacja takiego wyniku jest następująca:
- różnica dotyczy linii, która jest linią numer 2 w pliku lewym i która powinna się znaleźć **za** linią numer 1 w pliku prawym, aby pliki były identyczne; alternatywnie, linię tę należy usunąć z pliku lewego (*2d1*)
- poniżej zacytowano linię, której – z punktu widzenia polecenia **`diff`** – usunięcie sprawi, że pliki nie będą się różnić

2.5.11. Polecenie `cmp`

`cmp` ***`[przełącznik...] plik1 plik2`***

`cmp` porównuje zawartość dwóch plików dowolnego typu i wypisuje na konsolę wynik tego porównania (albo nie). Domyślnie **`cmp`** zachowuje pełne godności milczenie, gdy pliki są identyczne, a w przeciwnym przypadku **`cmp`** oznajmia numer bajtu i linii, w których wykryto pierwszą różnicę. *Uwaga: bajty i linie są numerowane od* ***jeden***.

Najprzydatniejsze przełączniki to:

-c wypisuje różniące się znaki

-i *n* ignoruje wszelkie różnice w *n* początkowych bajtach każdego z plików; traktuje pliki zawierające mniej niż *n* bajtów tak, jakby były puste

-l dla każdej różnicy wypisuje numer bajtu (dziesiętnie) i wartości różniących się bajtów (ósemkowo).

Do demonstracji działania polecenie **cmp** wykorzystamy dwa spreparowane pliki, różniące się dokładnie na dwóch bajtach. Utworzymy je posługując się sposobem zastosowanym wcześniej przy prezentacji polecenia **du**.

```
$ echo -n '0123456789' > plik1
$ echo -n '012x4567x9' > plik2
$ cmp plik1 plik2
plik1 plik2 różnią się: bajt 4, linia 1
$ cmp -l plik1 plik2
 4  63 170
 9  70 170
$ cmp -c plik1 plik2
plik1 plik2 różnią się: bajt 4, linia 1 zawiera  63 3 170 x
$ cmp -cl plik1 plik2
 4  63 3    170 x
 9  70 8    170 x
```

2.6. Źródła uzupełniające

1. Manual pliku **/etc/group**: `https://man7.org/linux/man-pages/man5/group.5.html`
2. Manual pliku **/etc/gshadow**: `https://man7.org/linux/man-pages/man5/gshadow.5.html`
3. Manual pliku **/etc/passwd**: `https://man7.org/linux/man-pages/man5/passwd.5.html`
4. Manual pliku **/etc/shadow**: `https://man7.org/linux/man-pages/man5/shadow.5.html`
5. Manual metadanych zagregowanych w i-węzłach: `https://man7.org/linux/man-pages/man7/inode.7.html`
6. Manual polecenia **chgrp**: `https://man7.org/linux/man-pages/man1/chgrp.1.html`
7. Manual polecenia **chmod**: `https://man7.org/linux/man-pages/man1/chmod.1.html`
8. Manual polecenia **chown**: `https://man7.org/linux/man-pages/man1/chown.1.html`

9. Manual polecenia **cmp**: https://man7.org/linux/man-pages/man1/cmp.1.html
10. Manual polecenia **date**: https://man7.org/linux/man-pages/man1/date.1.html
11. Manual polecenia **df**: https://man7.org/linux/man-pages/man1/df.1.html
12. Manual polecenia **diff**: https://man7.org/linux/man-pages/man1/diff.1.html
13. Manual polecenia **du**: https://man7.org/linux/man-pages/man1/du.1.html
14. Manual polecenia **echo**: https://man7.org/linux/man-pages/man1/echo.1.html
15. Manual polecenia **file**: https://www.man7.org/linux/man-pages/man1/file.1.html
16. Manual polecenia **ln**: https://man7.org/linux/man-pages/man1/ln.1.html
17. Manual polecenia **passwd**: https://man7.org/linux/man-pages/man1/passwd.1.html
18. Manual polecenia **stat**: https://man7.org/linux/man-pages/man1/stat.1.html
19. Manual polecenia **sync**: https://man7.org/linux/man-pages/man1/sync.1.html
20. Manual polecenia **truncate**: https://man7.org/linux/man-pages/man1/truncate.1.html

2.7. Zadania do samodzielnego wykonania

1. Czy kolejność zestawów *kto-jak-co*, wymienionych w jednym poleceniu **chmod** ma znaczenie? Podaj przykład, który uzasadnia twoją odpowiedź.
2. Oznacz jako **x** plik, który nie jest plikiem wykonywalnym i wykonaj go. Jak zareagował system operacyjny?
3. Zapoznaj się z zawartością pliku **/etc/passwd**. Czy są tam użytkownicy, który używają powłoki innej niż **/bin/bash**?
4. Spróbuj zapoznać się z zawartością pliku **/etc/shadow**. Co się stało i dlaczego?
5. Zapoznaj się z zawartością pliku **/etc/group**. Czy są tam grupy, do których należy więcej niż jeden użytkownik?
6. Odszukaj w systemie operacyjnym plik wykonywalny **busybox** i stwórz do niego link symboliczny o nazwie **cp**. Uruchom link symboliczny (upewnij się, że uruchamiasz **busyboksa**, a nie oryginalne polecenie **cp**). Sprawdź,

czy możesz po swoim dowiązaniu oczekiwać zachowania zgodnego z **cp**. Jakie jeszcze polecenia potrafi emulować **busybox**?

7. Czy przy pomocy polecenia **ln** można dowiązać się do któregoś ze swoich nadkatalogów i tym samy stworzyć zapętloną strukturę katalogów?
8. Stwórz plik, a następnie dowiązanie symboliczne, które prowadzi do niego. Usuń plik. Jak teraz zachowuje się dowiązanie?
9. Stwórz plik, a następnie dowiązanie do niego, a następnie dowiązanie do tego dowiązania. Zbadaj właściwości tego tworu. Spróbuj rozbudować łańcuch dowiązań.
10. Użyj polecenia **chmod** w celu zmiany praw dowiązania symbolicznego. Co się stało?
11. Użyj polecenia **stat** do zaprezentowania informacji o pliku w sposób maksymalnie zbliżony do wyjścia z polecenia **ls -l**
12. W systemie operacyjnym istnieje para programów nazywających się **true** i **false**. Zbadaj ich działanie i zapoznaj się z ich opisem w manualu. Jakiego skutku można oczekiwać, gdy w pliku **/etc/passwd** pewien użytkownik będzie miał jako swoją powłokę wpisany jeden z tych programów? Czy obecnie w twoim systemie jest taki użytkownik?
13. Przeanalizuj sposób, w jaki program **diff** prezentuje odnalezione różnice między plikami. Zaplanuj i przeprowadź stosowny eksperyment, posługując się kolejno plikami identycznymi, różniącymi się nieznaczne, różniącymi się znacznie i kompletnie innymi Upewnij się, że potrafisz przewidzieć, co powie ci **diff**, jeśli znasz dobrze zawartość obu plików.
14. Jeśli czasy w metadanych pliku byłyby reprezentowane przy pomocy liczb 32-bitowych, to kiedy skończyłaby się ich pojemność?
15. Czy polecenie **file** jest w stanie odróżnić kod źródłowy w C od kodu źródłowego w C++? Przeprowadź eksperyment.
16. Posługując się poleceniami **watch** i **date** skonstruuj prosty, odświeżany co sekundę zegarek, który prezentuje informację o czasie w sposób przedstawiony poniżej (ostatnia linia prezentuje sekundy):

```
22 październik
2014
13:44
00
```

17. Użyj polecenia **date** do wypisania daty w formie zgodnej z normą ISO 8601, czyli na przykład **2024-10-30**.
18. Który tydzień roku mamy obecnie według ustaleń normy ISO 8601?
19. Co to jest *rozmiar pozorny* prezentowany przez polecenie **du**?

20. Co pojawi się na ekranie, jeśli użyjesz **du** z przełącznikiem **-B512**?
21. Sprawdź, ile przestrzeni dyskowej zajmuj twój katalog domowy.
22. Wydaj z konsoli polecenie, które zasygnalizuje ci upływ jednej minuty.