Linux i Python w Elektronicznej Sieci #01: Wprowadzenie do Unix'a

Projekt „Matematyka dla Ciekawych Świata",
Robert Ryszard Paciorek
<rrp@opcode.eu.org>

2021-03-04

1 Praca w terminalu

Komputer potrafi jedynie wykonywać jakiś wcześniej zaprogramowany ciąg instrukcji. Każde wydane przez użytkownika polecenie wiąże się z uruchomieniem takiego ciągu instrukcji (programu komputerowego lub jakieś funkcji w ramach niego).

Podstawowym sposobem wydawania poleceń w systemach typu Unix jest wpisywanie ich w terminalu. Terminal może pracować w trybie tekstowym lub może być uruchomiony (jako tzw. emulator terminala) w trybie graficznym.

1.1 Powłoka

Wprowadzane polecenia interpretowane są przez działający w terminalu program nazywany powłoką (interpreterem poleceń). W terminalu mogą być uruchamiane kolejne (takie same lub różne) interpretery poleceń. Różne interpretery korzystają z różnych składni oraz często różnią się znakiem zachęty (czyli wypisanym tekstem poprzedzającym wprowadzane polecenia).

1.1.1 bash

Bash jest chyba najpopularniejszą powłoką (interpreterem poleceń) systemową w środowiskach linuksowych. Jest zgodny ze składnią z sh, zapewnia m.in. obsługę zmiennych (zasadniczo napisowych) oraz omówionych w dalszej części skryptu znaków uogólniających.

Edycja i historia linii poleceń

Bash, podobnie jak wiele innych interpreterów poleceń (np. Python) umożliwia edycję linii poleceń oraz korzystanie z jej historii.[a] Poruszanie się po historii linii poleceń możliwe jest strzałkami góra/dół, wyszukiwanie w historii przy pomocy Ctrl+R. Dostępne jest też polecenie wbudowane `history` umożliwiające wypisanie całej zapamiętanej historii oraz zarządzanie nią.
Bash pozwala także na dopełnianie nazw poleceń i ścieżek (a po odpowiedniej konfiguracji – pakiet *bash-completion* – także innych argumentów poleceń) przy pomocy tabulatora.

a. Bash korzysta w tym celu z popularnej biblioteki *GNU Readline* (więcej na jej temat: https://en.wikipedia.org/wiki/GNU_Readline). Biblioteka ta może zostać użyta w kodzie programu do obsługi wejścia, ale może zostać dodana do także zewnętrznie przy pomocy np. `rlwrap`.

1.1.2 screen i tmux

`screen` i `tmux` są tzw. multiplexerami terminala - pozwala na uzyskanie wielu okien konsoli (także np. wyświetlanych jedno obok drugiego) na pojedynczym terminalu. Ponadto pozwalają na odłączanie i podłączanie sesji, co pozwala na łatwe pozostawienie działającego programu po wylogowaniu i powrót do niego później.

1.2 Komendy

Unixowe komendy (czyli polecenia rozumiane przez bash lub inny interpreter zgodny z sh) składają się z nazwy polecenia oraz opcji i argumentów. Nazwą polecenia może być nazwa funkcji wbudowanej, nazwa programu (znajdującego się w ścieżce wyszukiwania programów) lub pełna ścieżka do programu. Po nazwie polecenia mogą występować opcje i/lub argumenty. Są one oddzielane od nazwy polecenia i od siebie przy pomocy spacji[1]. Nie ma silnego rozróżnienia opcji od argumentów, typowo stosowaną konwencją jest rozpoczynanie opcji od pojedynczego myślnika (opcje krótkie - jednoliterowe) lub dwóch myślników (opcje długie). W przypadku stosowania tej konwencji po pojedynczym myślniku może występować kilka bezargumentowych opcji jednoliterowych. Typowo argumenty opcji oddzielane są od nich spacją (w przypadku opcji krótkich) lub znakiem równości (w przypadku opcji długich). Jeżeli któryś z składników komendy (np. argument) zawiera spacje należy je zabezpieczyć przy pomocy odwrotnego ukośnika lub ujęcia zawierającego je napisu w apostrofy lub cudzysłowia.

1.3 Uzyskiwanie pomocy

Informację na temat działania danej komendy oraz jej opcji można uzyskać w wbudowanym systemie pomocy przy pomocy poleceń `man` lub `info` / `pinfo`. Większość poleceń obsługuje także opcje `--help` lub `-h`, które wyświetlają informację na temat ich użycia.

Notacja

Zarówno w tekstach pomocy jak i w tym dokumencie stosowana jest konwencja polegająca na oznaczaniu opcjonalnych argumentów poprzez umieszczanie ich w nawiasach kwadartowych (jeżeli podajemy ten argument do komendy nie obejmujemy go już tymi nawiasami) oraz rozdzielaniu alternatywnych opcji przy pomocy `|`. Np. `a [b] c|d` oznacza iż polecenie `a` wymaga argumentu postaci `c` albo `d`, który może być poprzedzony argumentem `b`.

1.4 more i less

Jeżeli wynik jakiejś komendy nie mieści się na ekranie do jego obejrzenia możemy użyć poleceń `more` lub `less`. Są to programy umożliwiające przeglądanie tekstu ekran po ekranie. `less` posiada większe możliwości od more (w szczególności posiada możliwość przeglądanie dokumentu w tył)[2]. Programy te kończą się po wciśnięciu klawisza `q`. `less` umożliwia także wyszukiwanie – klawisz `/` pozwala na wprowadzenie szukanej frazy, a `n` na wyszukanie kolejnego wystąpienia. Programy te umożliwiają też wyświetlanie wskazanych jako argumenty plików.

1.5 Przekierowania

Typowo program posiada trzy strumienie danych: jeden wejściowy (stdin) i dwa wyjściowe (stdout i stderr). Standardowe wyjście możemy przekierować na standardowe wejście innego programu przy pomocy `|`, np:

```
ls --help | less
```

Konstrukcja ta przekieruje wynik komendy `ls` uruchomionej z opcją `--help` do komendy `less`.

Możemy także przekierować standardowe wyjście do pliku (przy pomocy `>` lub `>>`, gdy chcemy dopisywać do pliku) lub pobrać standardowe wejście z pliku (przy pomocy `<`). `2>` pozwala na przekierowanie standardowego wyjścia błędu do pliku.

Jeżeli zachodzi potrzeba połączenia obu strumieni możemy użyć `2>&1` w celu przekierowania strumienia drugiego do pierwszego. Następnie możemy użyć `|` aby przekierować połączony strumień do następnej

1. zasadniczo dowolnego ciągu białych znaków: spacji, tabulatorów, „escapowanych” nowych linii.
2. wybrane przydatne opcje: `-X` nie czyści ekranu przy wychodzeniu z less'a (całość historii wyświetlania pliku pozostaje w historii terminala) `-F` automatycznie kończy gdy wyświetlany tekst mieści się na jednym ekranie

komendy. Jeżeli chcemy przekierować go do pliku połączenie strumieni powinno mieć miejsce po przekierowaniu pierwszego z nich do pliku, np.:

```
ls . NieIstniejącyPlik >log.txt 2>&1
```

Bash pozwala użyć `>&` i `|&`, które przekierowują oba strumienie odpowiednio do pliku lub standardowego wejścia innego polecenia, ale jest to rozszerzenie wykraczające poza standardową składnię sh.

1.6 Kod powrotu polecenia oraz łączenie poleceń

Każde uruchamiane polecenie po zakończeniu działania zwraca liczbowy kod powrotu (w przypadku programów w C jest to wartość zwracana z funkcji `main`). Zero oznacza że polecenie zakończyło się sukcesem (np. znaleziono szukane pliki), wartość nie zerowa że zakończyło się porażką (np. nie ma pasujących plików) lub błędem (np. składnia wprowadzonego polecenia była niepoprawna).

Polecenia mogą być łączone na różne sposoby – z wykorzystaniem tej informacji lub nie:

- `a && b` – polecenie b wykona się gdy a zakończyło się sukcesem (zwróciło kod 0)
- `a || b` – polecenie b wykona się gdy a zakończyło się porażką lub błędem (zwróciło kod różny od 0)
- `a ; b` – polecenie b po zakończeniu polecenia a (bez względu na jego kod powrotu)
- `a & b` – polecenie b będzie wykonywane równocześnie z a (dokładniej polecenie a zostanie uruchomione w tle, a na terminal zajmie polecenie b)

Spacje w powyższych konstrukcjach są opcjonalne. Średnik i pojedynczy `&` mogą być dodane do polecenia także gdy nie ma kolejnego w ciągu:

- `a&` uruchomi polecenie a w tle i odda linię poleceń,
- `a;` uruchomi polecenie a (dokładnie tak samo jakby nie było tego średnika).

1.7 Katalog roboczy

System plików ma strukturę hierarchiczną (drzewiastą) i rozpoczyna się w korzeniu oznaczanym ukośnikiem: `/`. Możliwe jest wyrażanie wszystkich ścieżek od korzenia, jednak nie jest to zbytnio wygodne. Interpreter poleceń taki jak *bash* potrafi znajdować się gdzieś w tej strukturze plików i miejsce to nazywane jest bieżącym katalogiem roboczym (*Present Working Directory*). Względem niego będą wyrażane ścieżki nie zaczynające się od korzenia, może być też oznaczony jawnie przy pomocy pojedynczej kropki.

- `cd [ścieżka]` zmiana bieżącego katalogu,
  warto zauważyć, iż katalogi w ścieżce oddzielamy ukośnikami `/`, bieżący katalog oznaczamy kropką `.`, nadrzędny oznaczamy dwiema kropkami `..`,
  ścieżki zaczynające się od ukośnika `/` oznaczają *ścieżki bezwzględne* (od korzenia systemu plików), pozostałe oznaczają *ścieżkę względną* (wyrażoną względem bierzącego katalogu),
  katalog domowy oznacza się tyldą `~`
- `pwd` wyświetla ścieżkę do bieżącego katalogu

1.8 vi i vim

`vi` jest chyba najbardziej zaawansowanym edytorem, którego obecność gwarantuje standard POSIX[3]. `vim` jest mocno rozbudowanym jego klonem, oferującym bardzo zaawansowane funkcjonalności, powszechnie stosowanym jako zamiennik oryginalnego `vi`. `vim` obsługuje 3 podstawowe tryby pracy: komend (służący do wydawania opisanych niżej poleceń), wizualny (służący do zaznaczania i wydawania niektórych komend), edycji (wstawiania/nadpisywania - służący do wprowadzania tekstu). Podstawowa klawiszologia:

- przełączanie pomiędzy trybami:
  - `Esc` powrót do trybu komend

3. IEEE Std 1003.1-2017 (The Open Group Base Specifications Issue 7, 2018 edition), XCU part https://pubs.opengroup.org/onlinepubs/9699919799/

  - `i` tryb wstawiania; `A` tryb wstawiania ze skokiem na koniec linii, `o` / `O` tryb wstawiania ze wstawieniem nowej linii po / przed bierzącą
  - `R` tryb zastępowania
  - `Insert` zmiana trybu wstawiania i zastępowania
  - `v` tryb wizualny (umożliwia zaznaczenie przy pomocy strzałek); `ctrl+v` tryb wizualny blokowy, `V` tryb wizualny liniowy
  - `:set paste` włącza `:set nopaste` wyłącza tryb wklejania (nie będzie działać automatyczne formatowanie itp.)
  - `gv` ponawia ostatnie zaznaczenie trybu wizualnego
- wycinanie i kopiowanie:
  - `y` skopiuj; `d` - wytnij (skopiuj i usuń) po `y`, `d` można podać np. `20l` lub `20[strzałka w prawo]` co oznacza 20 kolejnych znaków, `2w` oznacza dwa słowa (więcej o takich punktach skoku poniżej)
  - `x` wytnij (skopiuj i usuń) znak (może być poprzedzone ilością znaków do wycięcia); wielkie `X` działa analogicznie, tyle że w tył
  - `yy` skopiuj linię; `dd` - wytnij (skopiuj i usuń) w obu wypadkach może być poprzedzone ilością linii do skopiowania/wycięcia
  - `p` wkleja po; `P` - wkleja przed
  - komendy kopiowania i wklejania mogą być poprzedzone jedno-znakową nazwą rejestru w którym umieszczane są dane (poprzedzamy ją znakiem `"` i podajemy przed licznikiem, np. `"a3dd` wytnie do rejestru a 3 linie), część rejestrów jest używana automatycznie, a niektóre są tylko do odczytu, podgląd aktualnej zawartości rejestrów możliwy jest przy pomocy komendy `:registers`
- wyszukiwanie, zastępowanie, skok do linii:
  - `/` szukanie w przód, `?` szukanie w tył; `*` szukanie w przód słowa pod kursorem, `#` szukanie w tył słowa pod kursorem
  - `n` wyszukanie następnego wystąpienie; `N` wyszukanie poprzedniego wystąpienie
  - `G` przejście do wskazanej linii, numer podajemy przed G, 0 oznacza ostatnią linię w pliku, więc `0G` spowoduje przejście do niej
  - `:[zakres]s@regexp@napis@[g]` wyszukaj i zastąp wyrażenie regularne regexp przez napis; zakres może być:
    * numerem linii,
    * przedziałem z numerami linii postaci `pierwsza,ostatnia`, gdzie: `.` oznacza bieżącą linię, `$` oznacza ostatnią linię w pliku, wartość numeryczna poprzedzona `+` oznacza tyle kolejnych linii od bieżącej, a poprzedzona `-` przed bieżącą,
    * znakiem `%` (co oznacza cały plik),
    * zakresem zaznaczonym w trybie wizualnym;

    podanie opcji g powoduje zastępowanie wszystkich wystąpień a nie tylko pierwszego; znak `@` pełni rolę separatora i może zostać zamiast niego użyty inny znak
- otwieranie, zapisywanie, zamykanie plików:
  - `:e ścieżka` otwarcie wskazanego pliku
  - `:w` zapis (można także podać ścieżkę pod jaka ma zostać zapisany plik)
  - `:q` wyjście
  - `:q!` wyjście bez zapisywania
  - `:wq` zapis i wyjście
- przełączanie się między otwartymi plikami i oknami:

  - `:n` następny plik; `:N` poprzedni plik
  - `:split` poziomy podział okna; `:vs` pionowy podział okna; `Ctrl+W` a następnie strzałka - przełączanie między oknami
- cofanie i ponawianie edycji:
  - `u`, `:undo` cofa ostatnią operację
  - `Ctrl+r`, `:redo` ponawia cofniętą operację
- punkty skoku (mogą być używane jako polecenia do poruszania się lub jako adresy w poleceniach takich jak `d`, `y`):
  - `l` / `h` / `k` / `j` jeden znak/linię w prawo / lewo / górę / dół (działa tak jak strzałki)
  - `0` / `^` / `$` początek linii / początek tekstu w linii, koniec linii
  - `w` / `b` / `e` następne słowo / poprzednie słowo / koniec słowa; wielkie `W` / `B` / `E` działa analogicznie, różni się traktowaniem spacji przy słowie
  - `f` / `F` następny / poprzedni znak podany po tej komendzie, włącznie z nim (np. `dfX` usunie wszystko do najbliższego wystąpienia X wraz z tym X); `t` / `T` działa analogicznie, tyle wyłącza podany znak
  - poprzedzenie powyższych komend liczbą powoduje powtórzenie ich tyle razy - np. `10l` - 10 znaków w prawo, `3F:` - trzeci dwukropek w lewo
  - punktem skoku jest też wyżej opisane polecenie `G` poprzedzane numerem linii do której ma się odbyć skok
  - punktem skoku mogą być także swobodnie umieszczane z dokumencie zakładki identyfikowane pojedynczym znakiem:
    * `m` i następie znak ją identyfikujący - utworzenie zakładki w miejscu kursora (np. `ma` - utworzy zakładkę a)
    * `` ` `` (backtick) / `'` (apostrof) skok do zakładki / linii z zakładką podaną po tej komendzie
    * `:marks` - lista zakładek; `:delmarks` / `:delmarks!` - usunięcie zakładki / usunięcie wszystkich nie automatycznych zakładek
- inne:
  - `:r plik`, wstawienie zawartości pliku
  - `:%!xxd` pokazanie wartości numerycznych i umożliwienie edycji pliku jako binarnego; `:%!xxd -r` powrót do normalnej edycji
  - `>` / `<` zwiększanie / zmniejszanie wcięcia zaznaczonego (w trybie wizualnym) tekstu
  - `zc` zwija bieżący blok, `zC` zwija bieżący blok aż do najwyższego poziomu, `zo` rozwija bieżące zwinięcie, `zO` rozwija rekurencyjnie bieżące zwinięcie, `zR` rozwija wszystkie zwinięcia w dokumencie
  - `:set wrap` włącza `:set nowrap` wyłącza zawijania linii w podglądzie

2 Operacje na systemie plików

2.1 Podstawowe komendy

2.1.1 echo i znaki uogólniające

Polecenie echo służy do wypisania przekazanych do niego argumentów na ekran – np. `echo abc xyz` wypisze *abc xyz*. Polecenie to może zostać użyte także do wypisania plików pasujących do jakiegoś wzorca np. `echo a*` wypisze pliki zaczynające się literą *a*.

Dzieje się to dzięki obsłudze przez powłokę *znaków uogólniających*, które mogą być użyte w napisach i zostaną rozwinięte przez powłokę do listy pasujących ścieżek lub nazw plików. Podstawowymi znakami

uogólniającymi powłoki są:

- `?` oznaczający dowolny znak
- `*` oznaczający dowolny (także pusty) ciąg znaków
- `[a-z AD]` oznaczający dowolny znak z wymienionych w zbiorze ujętym w nawiasach kwadratowych, zbiór może być definiowany z użyciem zakresów, np. a-z AD oznacza dowolną małą literę od a do z włącznie, spację, dużą literą A lub D
- `[!a-z]` oznaczający dowolny znak z wyjątkiem znaków wymienionych w podanym zbiorze, zbiór może być definiowany z użyciem zakresów, np. a-z oznacza dowolną małą literę od a do z włącznie

Warto zwrócić uwagę że sama gwiazdka nie dopasowuje plików ukrytych (zaczynających się od kropki), czyli np. `echo *` wypisze wszystkie pliki w bieżącym katalogu, z wyjątkiem tych których pierwszym znakiem w nazwie jest kropka.

Napisy (a więc także ścieżki i nazwy plików) mogą być ujęte w cudzysłowie pojedynczym (', np. `'aaa bbb'`) lub podwójnym (", np. `"aaa bbb"`) celem np. ochrony spacji w nich występujących. Oba typy cudzysłowów zabezpieczają przed rozwijaniem znaków uogólniających (zastępowaniem napisu ze znakami listą pasujących nazw / ścieżek). Cudzysłów pojedynczy (w odróżnieniu od podwójnego) zabezpiecza także przed interpretacją umieszczonych wewnątrz innych znaków specjalnych takich jak odwołania do zmiennych.

Listowanie plików jest operacją na tyle istotną i użyteczną że istnieje do tego dedykowane polecenie – `ls`. Pozwala ono na m.in. wypisywanie szczegółowych informacji o plikach, listowanie całej zawartości katalogu, sortowanie wyników itd. Należy jednak pamiętać że rozwijaniem znaków uogólniających (czyli zamianą napisu je zawierającego na listę plików) dla polecenia `ls` zajmuje się powłoka (czyli np. `bash`) – tak samo jak dla polecenia `echo` – gdyż sama komenda `ls` nie rozumie znaków uogólniających. Kilka przykładów:

- pliki z jednoznakową nazwą: `ls ?`
- pliki zaczynające się od znaku zapytania: `ls \?*` lub `ls '?'*`
- pliki nie zaczynające się od a: `ls [!a]*`
- pliki mające w nazwie literę b: `ls *b*`
- pliki mające w nazwie literę a lub b: `ls *[ab]*`
- pliki zaczynające się od liter a,b,c,d: `ls [a-d]*`
- pliki ukryte (wliczając bieżący i nadrzędny katalog): `echo .*`

2.1.2 listowanie i wyszukiwanie plików

- `ls [opcje] [ścieżka]` listowanie zawartości katalogu, do ważniejszych opcji należy zaliczyć:
  `-a` wyświetlaj pliki ukryte (zaczynających się od kropki)
  `-l` wyświetlaj pliki w formie listy z szczegółowymi informacjami (uprawnienia, rozmiar, data modyfikacji, właściciel, grupa, rozmiar)
  `-1` wyświetlaj pliki w formie 1 plik w jednej linii (bez dodatkowych informacji; stosowane domyślne gdy wynik komendy przekazywany jest strumieniem do innej komendy lub pliku)
  `-h` stosuj jednostki typu k, M, G zamiast podawać rozmiar w bajtach
  `-t` sortuj wg daty modyfikacji
  `-S` sortuj wg rozmiaru
  `-r` odwróć kolejność sortowania
  `-c` użyj daty utworzenia zamiast daty modyfikacji (stosowane w połączeniu z `-l` i/lub `-t`)
  `-d` wyświetlaj informacje o katalogu zamiast jego zawartości
- `find [opcje] [katalog startowy] [wyrażenie]` wyszukiwanie w systemie plików w oparciu o nazwę/ścieżkę lub właściwości pliku, do ważniejszych opcji należy zaliczyć:
  `-P` wypisuj informacje o linkach symbolicznych a nie plikach przez nie wskazywanych (domyślne)
  `-L` wypisuj informacje o wskazywanych przez linki symboliczne plikach
  do ważniejszych elementów wyrażenia należy zaliczyć:

`-name "wyrażenie"` pliki których nazwa pasuje do wyrażenia korzystającego z shellowych znaków uogólniających
komenda find (w odróżnieniu np. od `ls`) samodzielnie interpretują wyrażenia zawierające shellowe znaki uogólniające, w związku z czym konieczne może się okazać zabezpieczenie ich przed interpretacją przez powłokę np. przy pomocy umieszczenia wewnątrz pojedynczych cudzysłowów
`-iname "wyrażenie"` jak `-name`, tyle że nie rozróżnia wielkości liter
`-path "wyrażenie"` pliki których ścieżka pasuje do wyrażenia korzystającego z shellowych znaków uogólniających
`-ipath "wyrażenie"` jak `-path`, tyle że nie rozróżnia wielkości liter
`-regex "wyrażenie"` pliki których ścieżka pasuje do wyrażenia regularnego
`-iregex "wyrażenie"` jak `-regexp`, tyle że nie rozróżnia wielkości liter

`warunek -o warunek` łączy warunki sumą logiczną „OR” (zamiast domyślnego iloczynu logicznego „AND”)
`! warunek` negacja warunku

`-mtime [+|-]n` pliki których modyfikacja odbyła się n*24 godziny temu
`-mmin [+|-]n` pliki których modyfikacja odbyła się n minut temu
`-ctime [+|-]n` pliki które zostały utworzone n*24 godziny temu
`-cmin [+|-]n` pliki które zostały utworzone n minut temu
`-size [+|-]n[c|k|M|G]` pliki których rozmiar wynosi n (c - bajtów, k - kilobajtów, M - Megabajtów, G - gigabajtów)
w powyższych testach `+` oznacza więcej niż, `-` oznacza mniej niż, uwaga: porównywaniu podlegają liczby całkowite, np. +1 oznacza > 1 *w liczbach całkowitych tzn.* $\geqslant 2$

`-exec polecenie \{\} \;` dla każdego znalezionego pliku wykonaj `polecenie` podstawiając ścieżkę do tego pliku pod `\{\}` (zastosowane odwrotne ukośniki służą zabezpieczeniu nawiasów klamrowych i średnika przed zinterpretowaniem ich przez powłokę)
`-execdir polecenie \{\} \;`, podobnie jak `-exec` tyle że polecenie zostanie uruchomione w katalogu w którym znajduje się wyszukany plik

- `du [opcje] ścieżka1 [ścieżka2 [...]]` wyświetlanie informacji o zajętej przestrzeni dyskowej przez wskazane pliki / katalogi, do ważniejszych opcji należy zaliczyć:
  `-s` podaje łączną ilość zajętego miejsca dla każdego argumentów (zamiast wypisywać rozmiar każdego pliku)
  `-c` podaje łączną ilość zajętego miejsca dla wszystkich argumentów
  `-h` stosuje jednostki typu k, M, G
  podawany rozmiar może się różnić (w obie strony) od wyniku ls: ls podaje rozmiar pliku (ile zawiera informacji lub ile zostało zadeklarowane że może jej zawierać), a du to ile zajmuje na dysku
- `df [opcje]` wyświetlanie informacji o zajętości miejsca na poszczególnych systemach plików

Należy zwrócić uwagę iż komenda `find` potrafi sama rozwijać znaki uogólniające[4] i w przypadku argumentów opcji takich jak np. `-name` na ogół chcemy aby znaki uogólniające nie były rozwijane przez powłokę, a interpretowane przez samą komendę `find` – w tym celu powinniśmy je zabezpieczyć przed rozwinięciem przy pomocy cudzysłowów.

Warto zauważyć także, że jeżeli komanda `ls` w wyniku rozwinięcia znaków uogólniających dostanie jako argument ścieżkę do katalogu to wylistuje jego zawartość (zachowanie to zmienia opcja `-d`).

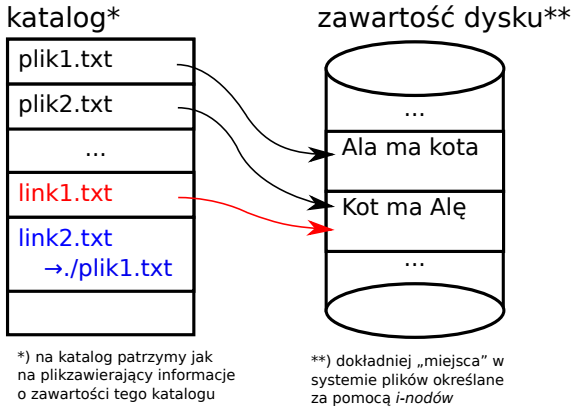
2.1.3 kopiowanie, przenoszenie, usuwanie, ...

- `cp [opcje] źródło1 [źródło2 [...]] cel` kopiuje wskazany plik (lub pliki) do wskazanej lokalizacji, w przypadku kopiowania wielu plików cel powinien być katalogiem, do ważniejszych opcji

4. W przypadku argumentów niektórych z jej opcji. W przypadku określania katalogu startowego `find` zachowuje się jak inne komendy (np. `ls`) dla których znaki uogólniające musi rozwinąć powłoka. Na przykład jeżeli chemy przeszukać wszystkie katalogi zaczynające się na *a* w poszukiwaniu plików zaczynających się na *b* to należy wykonać: `find a* -name "b*"`, a nie `find "a*" -name "b*"` czy `find a* -name b*`, itd.

należy zaliczyć:
`-r` pozwala na (rekursywne) kopiowanie katalogów
`-a` podobnie jak `-r`, dodatkowo zachowując atrybuty plików
`-l` zamiast kopiować tworzy twarde dowiązania (hard links)
`-s` zamiast kopiować tworzy linki symboliczne do plików
`-f` nadpisywanie bez pytania
`-i` zawsze pytaj przed nadpisaniem

- `ln źródło1 [źródło2 [...]] cel` tworzy link (domyślnie „twardy") do wskazanego pliku (lub plików) w wskazanej lokalizacji, w przypadku wskazania wielu plików źródłowych cel powinien być katalogiem, do ważniejszych opcji należy zaliczyć:
  `-s` tworzy dowiązania symboliczne (wskazujące na ścieżkę do oryginalnego pliku) zamiast twardych (wskazujących na te same dane co oryginalny plik)
  `-r` używa ścieżki względnej zamiast bezwzględnej przy tworzeniu dowiązań symbolicznych



`ln plik2.txt link1.txt`
powoduje utworzenie linku twardego `link1.txt`, czyli wpisu w katalogu wskazujacego na te same dane (ten sam i-node) co `plik2.txt`

`ln -s ./plik1.txt link2.txt`
powoduje utworzenie linku symbolicznego `link2.txt`, czyli wpisu w katalogu wskazującego na podaną ścieżkę (`./plik1.txt`)

Link twardy jest innym uchwytem do tych samych danych i może być używany także po skasowaniu oryginalnego pliku. Liczbę dowiązań do danego pliku pokazuje m.in. komenda `ls` z opcją `-l`. Nie można utworzyć linków twardych do katalogów, ani do plików na innym zasobie dyskowym (innym systemie plików).
Link symboliczny wskazuje na konkretną ścieżkę (względną lub bezwzględną – co może mieć znaczenie przy przenoszeniu takiego linku) do dowolnego (nawet nie istniejącego – wtedy mówimy o zerwanym linku) pliku lub katalogu.

- `mv [opcje] źródło1 [źródło2 [...]] cel` przenosi wskazane pliki / katalogi do wskazanej lokalizacji, w przypadku przenoszenia wielu plików cel powinien być katalogiem, do ważniejszych opcji należy zaliczyć:
  `-f` nadpisywanie bez pytania
  `-i` zawsze pytaj przed nadpisaniem
- `rm [opcje] ścieżka1 [ścieżka2 [...]]` usuwa wskazane pliki, do ważniejszych opcji należy zaliczyć:
  `-r` pozwala na na (rekursywne) kasowanie katalogów wraz z zawartością
  `-f` usuwanie bez pytania
  `-i` zawsze pytaj przed usunięciem
- `mkdir [opcje] ścieżka1 [ścieżka2 [...]]` tworzy wskazane katalogi, do ważniejszych opcji należy zaliczyć:
  `-p` pozwala na tworzenie całej ścieżki a nie tylko ostatniego elementu, nie zgłasza błędu gdy wskazany katalog istnieje

2.2 Struktura katalogów

Systemy unix'owe posiadają drzewiasty system plików zaczynający się w katalogu głównym oznaczanym przez ukośnik (`/`), w którym zamontowany jest główny system plików (rootfs), inne systemy plików mogą być montowane w kolejnych katalogach. Do najistotniejszych katalogów należy zaliczyć:

- `/bin` zawierający pliki wykonywalne podstawowych programów

- `/sbin` zawierający pliki wykonywalne podstawowych programów administracyjnych
- `/lib` zawierający pliki podstawowych bibliotek
- `/usr` zawierający oprogramowanie dodatkowe (wewnętrznie ma podobną strukturę do głównego - tzn. katalogi `/usr/bin`, `/usr/sbin`, `/usr/lib`, itd)
- `/etc` zawierający konfiguracje ogólnosystemowe
- `/var` zawierający dane programów i usług (takie jak kolejka poczty, harmonogramy zadań, bazy danych)
- `/home` zawierający katalogi domowe użytkowników (często montowany z innego systemu plików, dlatego też root ma swój katalog domowy w `/root`, aby był dostępny nawet gdy takie montowanie nie doszło do skutku)
- `/tmp` zawierający pliki tymczasowe (typowo czyszczony przy starcie systemu); w Linuxie występuje też `/run` przeznaczony do trzymania danych tymczasowych działających usług takich jak numery pid, blokady, itp
- `/dev` zawierający pliki reprezentujące urządzenia; w Linuxie występuje też `/sys` zawierający informacje i ustawienia dotyczące m.in. urządzeń
- `/proc` zawierający informacje o działających procesach (w Linuxie także interfejs konfiguracyjny dla wielu parametrów jądra)

Pliki i katalogi których nazwa rozpoczyna się od kropki traktowane są jako pliki ukryte.

Z punktu widzenia programisty czy też użytkownika (prawie) wszystko jest plikiem, których istnieją różne rodzaje (zwykły plik, katalog, urządzenie znakowe, urządzenie blokowe, link symboliczny, kolejka FIFO, ...); pewnym wyjątkiem są urządzenia sieciowe (które nie mają reprezentacji w systemie plików (ale gniazda związane z nawiązanymi połączeniami obsługuje się zasadniczo tak jak pliki).

3 Wykład wideo

- *Podstawy pracy w terminalu* – https://www.youtube.com/watch?v=IQpU7xZhpbY
- *Edytor vi i vim* – https://www.youtube.com/watch?v=ItKu1cNSttw
- *Listowanie i wyszukiwanie plików* – https://www.youtube.com/watch?v=hAXAKlNASPo
- *Kopiowanie, przenoszenie, usuwanie plików* – https://www.youtube.com/watch?v=BL-IRlDKeZ8
- *Struktura drzewa katalogów* – https://www.youtube.com/watch?v=xIROSkRqCJ0

4 Zadania

Zadanie 4.0.1

Przy pomocy edytora *vim* otwórz plik `/etc/passwd` i zastąp wszystkie wystąpienia `bin` przez `XYZ`. Nie zapisuj pliku.

Zadanie 4.0.2

Wyświetlić nazwy (mogą być wraz z pełną ścieżką) wszystkich plików i katalogów znajdujących się bezpośrednio w `/etc/` których druga litera to `a` natomiast trzecia to `p` lub `s`.
Wskazówka: to zadanie nie wymaga stosowania find.

Zadanie 4.0.3

Wyszukaj (rekurencyjnie) wszystkie pliki w katalogu `/etc/` zmodyfikowane w przeciągu ostatnich 48 godzin.

Zadanie 4.0.4

Utworzyć w katalogu `/tmp` linki symboliczne `ll1` i `ll2` wskazujące na `/etc/passwd` odpowiednio poprzez ścieżkę bezwzględną i względną.

Zadanie 4.0.5

Zmodyfikuj rozwiązanie zadania 4.0.3 tak aby wyświetlać szczegóły (w tym datę modyfikacji) dla wyszukanych plików.

5 Rozwiązania

Poniżej zamieszczone są przykładowe rozwiązania „głównych" zadań z tego skryptu wraz z komentarzami. Wiemy że zajrzenie do nich już przy pierwszej trudności jest kuszące, mimo to rekomendujemy przynajmniej podjąć uczciwą, co najmniej kilkunastominutową na każde z zadań, próbę rozwiązania tych zadania bez zaglądania do odpowiedzi.

Pamiętaj!: Samodzielne rozwiązanie problemu (wraz z wszystkimi trudnościami po drodze i popełnionymi błędami) jest dużo bardziej kształcące od nawet wielokrotnego przepisania gotowego rozwiązania, jednak nawet jednokrotne przepisanie rozwiązania jest bardziej kształcące od wielokrotnego przekopiowania go.

Rozwiązanie zadania 4.0.1

1. Uruchomić edytor za pomocą polecenia `vim /etc/passwd`.
2. W uruchomionym edytorze należy wydać polecenie `:%s@false@FALSE@g`, które spowoduje zastąpienie wszystkich wystąpień `false` przez `FALSE`.
3. Edytor należy opuścić bez zapisywania zmian przy pomocy polecenia `:q!`

Zwróć uwagę że:

- Plik do otwarcia wskazujemy w linii poleceń przy pomocy ścieżki bezwzględnej zaczynającej się od `/`. Zamiast tego można by także otworzyć plik po uruchomieniu edytora lub użyć ścieżki względnej. Użycie ścieżki bezwzględnej zapewnia niezależność tej komendy od katalogu roboczego w którym zostanie wykonana.
- Do zamiany i zamknięcia edytora używane są polecenia linii poleceń vim'a uruchamianej przy pomocy dwukropka (`:`) i dlatego są one poprzedzane dwukropkiem. Jest to inny zbiór komend niż komendy wprowadzane bezpośrednio (takie jak np. `dd` kasujące bieżącą linię). Ta linia poleceń też ma swoją historię.
- Edytor opuszczamy z użyciem `:q!`, dodanie wykrzyknika jest potrzebne żeby zamknąć edytor bez zapisywania zmian.

Rozwiązanie zadania 4.0.2

```
ls -d /etc/?a[sp]*
```

albo:

```
ls -d /etc/?as* /etc/?ap*
```

Zwróć uwagę że:

- Polecenie `ls` listuje wszystkie podane w linii poleceń argumenty oddzielane spacjami.
- Jeżeli argumenty te zawierają znaki ogulniające powłoki to każdy taki argument zostanie zastąpiony przez powłokę listą pasujących plików, lub zostanie przekazany w formie niezmienionej jeżeli nie ma pasujących plików. Ma to kilka konsekwencji:
  - W wyniku dopasowania nazw do polecenia ls mogą zostać przekazane ścieżki do katalogów, dla których ls domyślnie listuje ich zawartość. Tutaj tego nie chemy i dlatego używamy opcji `-d`

  - W pierwszym wariancie wszystkie pliki opisaliśmy pojedynczym wyrażeniem uogólniającym basha, a w drugim podaliśmy dwa argumenty. Polecenia te są prawie równoważne - wypiszą te ścieżki. Jednak w przypadku braku dopsaowań do jednego z wariantów (np. braku plików z trzecią literą p) – drugie polecenie wypisałoby dodatkowo komunikat o braku dopasowania do jednego z podanych argumentów (np. do `/etc/?ap*`). Wynika to z tego iż przy braku dopsowania do tego argumentu bash przekazałby go w formie niezmienionej do polecenia `ls`.

Rozwiązanie zadania 4.0.3

```
cd /etc
find -mtime -2
```

lub

```
find /etc -mtime -2
```

jeżeli nie uważamy katalogów za pliki, to możemy dodać stosowny warunek

```
find -mtime -2 -type f
```

Zwróć uwagę że:

- polecenie find przeszukuje rekurencyjnie katalog podany przed warunkami. Jeżeli nie został on podany to przeszukuje bieżący katalog (`.`), który mozemy zmienić poleceniem `cd`.

Rozwiązanie zadania 4.0.4

Podając pełne ścieżki do pliku docelowego:

```
ln -s /etc/passwd /tmp/ll1
ln -s -r /etc/passwd /tmp/ll2
```

lub wchodząc do katalogu docelowego i podając tylko nazwy plików

```
cd /tmp
ln -s /etc/passwd ll1
ln -s ../etc/passwd ll2
```

Link ze ścieżką względna można utworzyć także będąc w dowolnym katalogu poprzez wywołanie:

```
ln -s ../etc/passwd ll2
```

Zwróć uwagę że:

- Pierwszy wariant możemy wykonać także będąc w `/tmp`
- W przypadku podawania w `ln` ścieżki względnej (`ln -s ../etc/passwd ll2`) należy pamiętać iż zostanie ona literalnie zapisana w tworzonym linku.
  - W związku z tym musi to być poprawna ścieżka z katalogu docelowego, a nie z bieżącego katalogu roboczego.
  - Np. będąc w `/usr/local/bin/` ścieżka względna do `/etc/passwd` to `../../../etc/passwd`, ale gdy będąc w tamtym katalogu chemy utworzyć plik `/tmp/llX` linkujący ścieżką względną do `/etc/passwd` wykonamy polecenie `ln -s ../etc/passwd /tmp/llX`, bo ścieżka względem `/tmp` ma postać `ln -s ../etc/passwd`.

Rozwiązanie zadania 4.0.5

```
find /etc -mtime -2 -exec ls -ld \{\} \;
```

Zwróć uwagę że:

- Wykorzystanie warunku `-exec`, który dodatkowo wyłącza standardowy output find'a
- Zabezpieczamy klamerki (nie zawsze wymagane) i średnik (praktycznie zawsze wymagane) oraz spację pomiędzy klamerkami a średnikiem. Możemy je zabezpieczyć także cudzysłowami: `find /etc -mtime -2 -exec ls -ld '{}' ';'`, ale nie możemy umieścić klamerek i średnika w jednym napisie (np. `find -mtime -2 -exec ls -ld '{\} ;'`).