4. Wyszukiwanie informacji

4.1. Wyrażenia regularne

Wyrażeniem regularnym (ang. *regular expression*, często skracane do *regex*, *regexp* albo *re*) nazywa się ciąg znaków (napis), tworzący wzorzec wyszukiwania (często również wzorzec zamiany), nakładany na inny ciąg w celu stwierdzenia istnienia lub braku zgodności między nimi. Pomysł wyrażeń regularnych pochodzi od amerykańskiego matematyka, Stephena Kleene'a (1909-1994), który w roku 1950 pracował nad przetwarzaniem i rozpoznawaniem języka naturalnego.

Każdy znak wyrażenia regularnego jest albo tak zwanym *metaznakiem* (któremu przypisuje się specjalne znaczenie) albo *literałem* (to znaczy znakiem, który oznacza sam siebie).

Semantykę wyrażeń regularnych definiuje się najczęściej przy pomocy tak zwanej *maszyny wyrażeń regularnych*, będącej deterministycznym automatem skończonym. Maszyna próbuje odszukać w przekazanym jej łańcuchu taką sekwencję znaków, która dopasowuje się do podanego jej wzorca. Wynik próby dopasowania zwracany jest jako rezultat działania maszyny.

Współcześnie wyrażenia regularne są powszechnie używane do wyszukiwania napisów i manipulowania nimi, przy czym podejście do ich stosowania bywa bardzo różnorodne. W niektórych językach programowania wyrażenia regularne

Rys. 4..1: Środowisko działania maszyny wyrażeń regularnych

są dostępne wprost i stanowią część definicji języka (na przykład Perl, AWK), w innych dostępne są za pośrednictwem funkcji bibliotecznych (na przykład język C i biblioteka *regex*) bądź specjalizowanych klas (na przykład Java i klasy *Pattern* i *Matcher*) czy też modułów (na przykład Python z modułem *re*).

Poniżej podajemy podzbiór metaznaków języka wyrażeń regularnych oraz reguły ich dopasowywania w postaci akceptowanej przez większość narzędzi w systemach uniksowych. W większości przypadków podajemy także przykłady wzorców i napisów wraz z wynikiem dopasowania

4.2. Reguły dopasowania wzorca

1. Dopasowanie wzorca do łańcucha odbywa się zawsze od lewej do prawej
2. Jeżeli pewien znak wzorca nie jest metaznakiem, to dopasowuje się do napisu literalnie, tzn. dopasowanie wystąpi tylko wtedy, gdy na odpowiednim miejscu napisu znajdować się będzie dokładnie taki sam znak (ewentualnie z dokładnością do wielkości litery); w szczególności wszystkie cyfry i litery dopasowują się zawsze literalnie.

napis	*wzorzec*	*dopasowanie?*
`Ala ma kota`	`kota`	tak
`Ala ma kota`	`koty`	nie

3. Znaki literalne można zapisywać za pomocą ich kodów ASCII, wyrażonych szesnastkowo z przedrostkiem code\x lub ósemkowo z przedrostkiem code\o (kod ASCII litery `A` to 41 szesnastkowo i 101 ósemkowo).

napis	*wzorzec*	*dopasowanie?*
`Ala ma kota`	`\x41`	tak
`Ala ma kota`	`\o101`	tak

4. Metaznakami są znaki z następującego zbioru:
   - `.` (kropka)
   - `^` (*caret*)
   - `$` (dolar)
   - `*` (gwiazdka)
   - `[]` (nawiasy kwadratowe)
   - `<>` (nawiasy ostre)
   - `()` (nawiasy okrągłe)
   - `\` (*backslash*)
5. Jeżeli za znakiem `\` stoi jeden ze znaków wymienionych poniżej, to element taki dopasowuje się do jednego ze znaków sterujących:
   - `\n` znak końca linii (*LF*)
   - `\r` znak końca linii (*CR*)
   - `\t` tabulator (*HT*)
   - `\e` tak zwany *znak ucieczki* (ang. *escape char* – w ASCII jest to znak o kodzie 27 i używany jest, między innymi, do sterowania terminalem)
   - `\f` znak przejścia do nowej strony (*FF*)
6. Jeżeli za znakiem `\` znajduje się jeden ze znaków wymienionych poniżej, to element taki dopasowuje się do klasy znaków (zauważ, że litera wielka neguje znaczenie litery małej):
   - `\s` tak zwany *biały znak* (ang. *whitespace*)
   - `\S` znak niebędący białym znakiem
   - `\w` znak w słowie (tak zwany *znak słowotwórczy* – litera, cyfra, podkreślenie)
   - `\W` znak, który nie jest znakiem w słowie
   - `\b` miejsce w napisie na granicy słowa
   - `\B` miejsce w napisie pomiędzy znakami słowa

napis	*wzorzec*	*dopasowanie?*
`Ala ma kota`	`\s\S\S\s`	tak
`Ala nie ma kota`	`e\b`	tak
`Ala nie ma kota`	`\be`	nie

7. Metaznak `.` (kropka) dopasowuje się do dowolnego (jednego!) znaku w

napisie:

napis	*wzorzec*	*dopasowanie?*
`Ala ma kota`	`kot.`	tak
`Ala ma koty`	`kot.`	tak

8. Ujęcie listy znaków w nawiasy kwadratowe `[]` definiuje tak zwany *zbiór znaków*; zbiór znaków dopasowuje się w pewnego znaku w napisie, gdy znak ten jest elementem zbioru:

napis	*wzorzec*	*dopasowanie?*
`Ala ma trzy koty`	`kot[uy]`	tak
`Ala idzie z kotem`	`kot[uy]`	nie
`Ala mówi kotu`	`kot[uy]`	tak

9. Jeżeli pierwszym znakiem zbioru jest `^`, to zbiór rozumiany jest jako swoje dopełnienie, tzn. traktowany jest tak, jakby składał się ze wszystkich tych znaków, których nie wymieniono:

napis	*wzorzec*	*dopasowanie?*
`Ala ma trzy koty`	`kot[^uy]`	nie
`Ala idzie z kotem`	`kot[^uy]`	tak
`Ala mówi kotu`	`kot[^uy]`	nie

10. Zbiór może być definiowany tak zwanym *zakresem*, podawanym jako krańce dolny i górny, rozdzielone znakiem łącznika (`-`); zakres rozumiany jest jako wszystkie znaki, które w danym alfabecie leżą pomiędzy wskazanymi krańcami wraz z tymi krańcami; jeżeli system operacyjny został prawidłowo zlokalizowany, zakres powinien obejmować sobą również ewentualnie istniejące pomiędzy krańcami znaki narodowe.

napis	*wzorzec*	*dopasowanie?*
`Ala ma 12 kotów`	`1[1-5]`	tak
`Ala ma 16 kotów`	`1[1-5]`	nie
`Ala ma 10 kotów`	`1[1-5]`	nie

11. Jeden zbiór może zawierać dowolnie wiele zakresów, a zakresy mogą być rozdzielane pojedynczymi elementami:

napis	*wzorzec*	*dopasowanie?*
`x = 0xff;`	`0x[0-9a-f]`	tak
`x = 0xff;`	`0x[0-9A-F]`	nie
`x = 0xff;`	`[0-9abc-f]`	tak

12. Jeżeli wewnątrz zbioru umieszczony jest element postaci `[:klasa:]`, to element taki dopasowuje się do odpowiedniego znaku w napisie wtedy, gdy znak ten należy do wskazanej klasy; definiuje się następujące klasy i ich nazwy:

nazwa	*klasa*	*zbiór równoważny*
`[:alnum:]`	litera (mała lub wielka) lub cyfra dziesiętna	`[a-zA-Z0-9]`
`[:alpha:]`	litera (mała lub wielka)	`[a-zA-Z]`
`[:ascii:]`	znak kodu ASCII	`[\x00-\x7F]`
`[:blank:]`	spacja lub tabulator	`[ \t]`
`[:cntrl:]`	znak sterujący	`[\x00-\x1F\7F]`
`[:digit:]`	cyfra dziesiętna	`[0-9]`
`[:lower:]`	mała litera	`[a-z]`
`[:graph:]`	znak widoczny (każdy oprócz białych i sterujących)	`[\x21-\x7E]`
`[:print:]`	znak drukowalny (każdy oprócz sterujących)	`[\x20-\x7E]`
`[:punct:]`	znak przestankowy	`[!"#$%&'()*+ ,\-./:;<=>?@ []^_‘{|}~]`
`[:space:]`	biały znak	`[ \t\r\n\f]`
`[:upper:]`	wielka litera	`[A-Z]`
`[:xdigit:]`	cyfra szesnastkowa	`[0-9a-fA-F]`

Na przykład:

napis	*wzorzec*	*dopasowanie?*
`Ola`	`[[:upper:]]la`	tak
`bla`	`[[:upper:]]la`	nie
`dla`	`[[:lower:]]la`	tak

13. Metaznak `^` dopasowuje się nie do znaku, a do miejsca, które leży przed

pierwszym znakiem napisu:

napis	*wzorzec*	*dopasowanie?*
`Ala ma kota`	`^Ala`	tak
`Ala ma kota`	`^kota`	nie

14. Metaznak `$` dopasowuje się nie do znaku, a do miejsca, które leży za ostatnim znakiem napisu:

napis	*wzorzec*	*dopasowanie?*
`Ala ma kota`	`Ala$`	nie
`Ala ma kota`	`kota$`	tak

15. Metaznaki `\<` i `\>` dopasowują się do miejsc, które leżą odpowiednio przed pierwszym i za ostatnim znakiem słowa (tzn. na początkowej i końcowej granicy słowa):

napis	*wzorzec*	*dopasowanie?*
`To jest rzeka`	`\<rzeka\>`	tak
`On narzekał`	`\<rzeka\>`	nie

16. Metaznak `?` oznacza, że poprzedni element wzorca występuje w napisie opcjonalnie (dokładnie zero razy lub raz):

napis	*wzorzec*	*dopasowanie?*
`To jest kot`	`koty?`	tak
`To są koty`	`koty?`	tak

17. Metaznak `*` oznacza, że poprzedni element wzorca może wystąpić w napisie dowolną liczbę razy (w tym zero razy):

napis	*wzorzec*	*dopasowanie?*
`Oto__kot`	`Oto_*kot`	tak
`Otokot`	`Oto_*kot`	tak

18. Metaznak `+` oznacza, że poprzedni element wzorca występuje w napisie co najmniej raz:

napis	*wzorzec*	*dopasowanie?*
`Oto__kot`	`Oto_+kot`	tak
`Otokot`	`Oto_+kot`	nie

19. Element postaci `{n}` gdzie *n* jest liczbą naturalną, oznacza, że poprzedni element wzorca występuje w napisie dokładnie *n* razy:

napis	*wzorzec*	*dopasowanie?*
`12345`	`[[:digit:]]{5}`	tak
`1234`	`[[:digit:]]{5}`	nie

20. Element postaci `{n,m}` gdzie *n* i *m* są liczbami naturalnymi oraz $n \leq m$ oznacza, że poprzedni element wzorca występuje w napisie co najmniej *n* razy i nie więcej niż *m* razy.

napis	*wzorzec*	*dopasowanie?*
`1`	`[[:digit:]]{2,3}`	nie
`1234`	`[[:digit:]]{2,3}`	tak

21. Element postaci `{n,}` gdzie *n* jest liczbą naturalną oznacza, że poprzedni element wzorca występuje w napisie co najmniej *n* razy.

napis	*wzorzec*	*dopasowanie?*
`1`	`[[:digit:]]{2,}`	nie
`123`	`[[:digit:]]{2,}`	tak

22. Metaznak `|` jest operatorem alternatywy i oznacza, że w napisie ma wystąpić jeden z elementów stojących na lewo i prawo od tego operatora.

napis	*wzorzec*	*dopasowanie?*
`Ala`	`O|Ala`	tak
`Ula`	`O|Ala`	nie

23. Metaznaki `(` i `)` służą do grupowania elementów wzorca i pełnią rolę podobną do tej, w jakiej używa się ich w wyrażeniach algebraicznych.

napis	*wzorzec*	*dopasowanie?*
`tato ma kota`	`(ma){2}`	nie
`mama ma kota`	`(ma){2}`	tak

4.3. Polecenie `grep`

Niektóre cechy polecenia **grep** poznałeś już w poprzednim rozdziale, kiedy to używaliśmy go w charakterze filtra. Teraz zapoznamy się z jego kolejnymi, bardziej zaawansowanymi możliwościami. Należy tu zaznaczyć, że historycznie **grep** występował w kilku odmianach różniących się sposobem działania – były to:

- **grep** – wersja podstawowa
- **egrep** – wersja akceptująca wyrażenia regularne
- **fgrep** – wersja akceptująca wzorce wyszukiwania podawane jako kolejne wiersze

Obecnie wszystkie te zadania wykonuje jedno narzędzie (**grep**), uruchamiane z odpowiednimi przełącznikami (na przykład **-E** włącza tryb **egrep**), jednak dla zachowania zgodności ze starymi konwencjami w systemie istnieją zwykle również pozostałe warianty.
Polecenie **grep**, oprócz pracy w potoku, kiedy to analizuje przepływający przez nie strumień znaków, potrafi przeglądać wskazane w argumentach pliki i dokonywać w nich przeszukań z wykorzystaniem wyrażeń regularnych. W trybie domyślnym **grep** wyprowadza na swoje standardowe wyjście wszystkie linie plików wejściowych, które albo zawierają poszukiwany ciąg znaków, albo dają się dopasować do wskazanego wyrażenia regularnego. W takim wariancie **grep** uruchamiany jest w następujący sposób:

grep ***[przełącznik...] wzorzec [plik...]***
gdy wzorzec jest prostym ciągiem znaków

grep -E ***[przełącznik...] wzorzec [plik...]***
gdy wzorzec jest wyrażeniem regularnym

Najwygodniejszym sposobem zapisu wzorca jest ujęcie go w apostrofy bądź cudzysłowy. W przeciwnym przypadku należy się liczyć z tym, że niektóre z metaznaków wyrażeń regularnych mogą być fałszywie interpretowane przez po-

włokę i wtedy konieczne będzie ich cytowanie za pomocą znaku \ (niezależnie od cytowania wymuszanego w wyrażeniach regularnych).

Oprócz przełączników opisanych w poprzednim rozdziale akceptowane są również poniższe:

-f *plik*
pobierz wzorce nie z linii poleceń, a z pliku o wskazanej nazwie

-x
wybieraj tylko te linie, które w całości dopasowują się do wzorca

-L
zamiast domyślnej postaci wyjścia wygeneruj listę nazw plików, z których nie wyprowadzono by żadnej pasującej linii

-l
zamiast domyślnej postaci wyjścia wygeneruj listę nazw plików, z których wyprowadzono by pasujące linie

-m *num*
przerwij czytanie pliku wejściowego po znalezieniu num dopasowań

-o
wypisuj tylko te części linii, dla których znaleziono dopasowanie (normalnie wypisywane są całe linie)

-s
wyłącz komunikaty o błędach w dostępie do przeglądanych plików

-b
przed każdą prezentowaną linią wyprowadź licznik bajtów prezentujący odległość linii od początku pliku

-H
przed każdą linią wyprowadź nazwę zawierającego ją pliku

-h
nie wyprowadzaj nazw plików (zachowanie domyślne, gdy przeszukiwany jest tylko jeden plik)

`-n`
przed każdą prezentowaną linią wyprowadź licznik linii

`-a`
przetwarzaj pliki binarne tak, jakby zawierały tekst

`--exclude=`*`wzorzec_nazwy`*
pomijaj pliki, których nazwy pasują do wzorca *wzorzec_nazwy* (*uwaga – to nie jest wyrażenie regularne, a prosta maska nazwy pliku ze znakami* **`?`** *i* **`*`** *w tradycyjnym znaczeniu*)

`--exclude-dir=`*`wzorzec_nazwy`*
pomijaj katalogi, których nazwy pasują do wzorca *wzorzec_nazwy* (*uwaga jak powyżej*)

`-r`
przetwarzaj katalogi rekursywnie; honoruj linki symboliczne tylko wtedy, gdy zostały jawnie wymienione w linii poleceń

`-R`
przetwarzaj katalogi rekursywnie; honoruj wszystkie linki symboliczne

`-w`
szukaj tylko linii, w których wzorzec dopasowuje się do całych słów

`-v`
wyprowadzaj tylko te linie, które **nie pasują** do wzorca

4.4. Przykłady użycia narzędzia `grep`

W zaprezentowanych poniżej przykładach zakładamy istnienie plików o nazwach **`test1`** i **`test2`** o następującej zawartości:

```
SAME WIELKIE LITERY
same małe litery
Linia Wielbłądzia Czyli Kapitaliki
```

```
powyżej są 2 puste linie
ostatnia linia
```

Poniżej podajemy kilka przykładów użycia polecenia **grep**, prezentując kompletne polecenia i dane wyprowadzane na standardowe wyjście w wyniku działania tego polecenia.

– `grep "litery" test1`

```
same małe litery
```

– `grep "li" test1`

```
same małe litery
Linia Wielbłądzia Czyli Kapitaliki
powyżej są 2 puste linie
ostatnia linia
```

– `grep "litery" test*`

```
test1:same małe litery
test2:same małe litery
```

– `grep -i "litery" test1`

```
SAME WIELKIE LITERY
same małe litery
```

– `grep -E "lini." test1`

```
powyżej są 2 puste linie
ostatnia l
```

– `grep -E "[[:digit:]]" test1`

```
  powyżej są 2 puste linie

– grep -E "^$" test1

  --pusta linia --
  --pusta linia --

– grep -v -E "^$" test1

  SAME WIELKIE LITERY
  same małe litery
  Linia Wielbłądzia Czyli Kapitaliki
  powyżej są 2 puste linie
  ostatnia linia

– grep -c -v -E "^$" test1

  5

– grep -l -i "li" test*

  test1
  test2

– grep -b -i "li" test*

  test1:0:SAME WIELKIE LITERY
  test1:20:same małe litery
  test1:38:Linia Wielbłądzia Czyli Kapitaliki
  test1:77:powyżej są 2 puste linie
  test1:104:ostatnia linia
  test2:0:SAME WIELKIE LITERY
  test2:20:same małe litery
  test2:38:Linia Wielbłądzia Czyli Kapitaliki
  test2:77:powyżej są 2 puste linie
  test2:104:ostatnia linia
```

4.5. Polecenie `find`

Polecenie **find** służy do wyszukiwania plików, których metadane spełniają określone kryteria. Składnia polecenia prezentuje się następująco:

```
find [-H] [-L] [-P] [katalog...] [wyrażenie]
```

Polecenie **find** będzie szukać plików o cechach określonych przez *wyrażenie*, a proces szukania będzie kolejno rozpoczynany w każdym z wymienionych *katalogów*.

Pierwsze trzy opcje, z których jednocześnie może wystąpić co najwyżej jedna (jeśli została podana, musi poprzedzać jakiekolwiek inne elementy linii poleceń) determinują sposób, w jaki **find** traktuje dowiązania symboliczne:

- **-H** ignoruj dowiązania symboliczne (zachowanie domyślne)
- **-L** honoruj dowiązania symboliczne, traktując je tak, jakby były obiektami, na które wskazują
- **-P** honoruj dowiązanie symboliczne tylko wtedy, gdy zostały jawnie wymienione w linii poleceń

Wyrażenie jest kombinacją warunków domyślnie połączonych spójnikiem „i”. Poleenie **find** umożliwia określenie innego sposobu wiązania warunków za pomocą nawiasów okrągłych i jawnych operatorów logicznych. Po znalezieniu pliku spełniającego podane warunki **find** wypisuje jego nazwę na swoje standardowe wyjście. Dodatkowo, w takiej sytuacji **find** może wykonać pewną akcję (na przykład uruchomić wskazane polecenie). Argumenty numeryczne będące składnikami warunków zapisuje się w sposób następujący:

- ***+n*** większe od *n*
- ***-n*** mniejsze od *n*
- ***n*** równe *n*

Warunki, odwołujące się do metadanych pliku określających czas, mają przedrostek:

- **a** gdy chodzi o *atime* (czas dostępu)
- **c** gdy chodzi o *ctime* (czas utworzenia)
- **m** gdy chodzi o *mtime* (czas modyfikacji)

Poniżej prezentujemy wybrane warunki sprawdzane przez polecenie **find**.

-min *n* (gdzie *min* to jedno z: **amin**, **cmin**, **mmin**)
czas *min* jest odległy o *n* minut od bieżącego
-newer *nazwa_pliku* (gdzie *newer* to jedno z: **anewer**, **cnewer**, **mnewer**)
szukany jest plik nowszy niż plik o nazwie *nazwa_pliku*
-time *n* (gdzie *time* to jedno z: **atime**, **ctime**, **mtime**)

czas *time* jest odległy od bieżącego o $n \cdot 24$ godziny (stosuje się zaokrąglenie w górę, które oznacza, że na przykład **`-atime +1`** opisuje plik, który był użyty przed dwoma dniami albo wcześniej)

`-empty`
szukany plik/katalog ma być pusty

`-executable`
szukany plik ma być wykonywalny lub katalog ma być „*przeglądalny*”

`-gid n`
plik ma należeć do grupy o numerze *n*

`-group g`
plik ma należeć do grupy o nazwie *g*

`-name nazwa`
nazwa odnalezionego pliku ma pasować do wzorca *nazwa* (wielkość liter **ma znaczenie**)

`-iname nazwa`
nazwa odnalezionego pliku ma pasować do wzorca *nazwa* (wielkość liter **nie ma znaczenia**)

`-perm prawa_pliku`
odnaleziony plik ma mieć prawa jak określone ósemkowo w *prawa_pliku*

`-readable`
szukany plik ma być odczytywalny

`-size n[cwbkMG]`
odnaleziony plik ma mieć rozmiar n jednostek, gdzie:

- **`b`** blok 512-bajtowy
- **`c`** bajt
- **`w`** słowo (2 bajty)
- **`k`** kibibajt
- **`M`** mebibaft
- **`G`** gibibajt

`-type t`
odnaleziony plik ma być:

- **`d`** katalogiem
- **`f`** zwykłym plikiem
- **`l`** dowiązaniem symbolicznym

`-uid n`
plik ma należeć do użytkownika o numerze *n*

`-user u`
plik ma należeć do użytkownika o nazwie *u*

`-writable`

plik ma być zapisywalny

W tym miejscu nieodzownym staje się uzupełnienie powyższego opisu o pewien szczegół dotyczący operatora **-size** użytego w połączeniu z rozmiarem poprzedzonym znakiem minus, a więc sprawdzającego, czy rozmiar pliku jest mniejszy niż żądany. Otóż z powodów znanych tylko i wyłącznie autorom polecenia **find** narzędzie to w czasie pracy zaokrągla rozmiary badanych plików w górę do takiej jednostki, która została użyty w operatorze. W efekcie rezultaty wyszukiwania w sposób absolutnie zaskakujący i kompletnie absurdalny (*sic!*) odbiegają od spodziewanych.

Załóżmy, że zawartość pewnego katalogu prezentuje się następująco:

```
-rw-r--r-- 1 user user    0 09-21 15:37 0
-rw-r--r-- 1 user user 1024 09-21 15:38 1024
-rw-r--r-- 1 user user    8 09-21 15:38 8
```

Zauważ, że nazwy plików są tożsame z ich rozmiarem w bajtach. W tak przygotowanym środowisku spróbujemy teraz znaleźć pliki mniejsze od 1 KiB.

```
$ find -size -1k
./0
```

Jak widać, **find** odnalazł plik **0** (poprawnie), ale nie odnalazł pliku **8**. Czemu? Wyjaśnienie tego faktu brzmi schizofrenicznie, ale jest prawdziwe:

- do wyszukiwania użyto jednostki **k**, więc rozmiary badanych plików będą zokrąglane do 1KiB
- rozmiar wyrażony jako 0 bajtów zokrąglony do 1KiB to nadal 0, więc plik zostaje uwzględniony w wynikach
- rozmiar wyrażony jako 8 bajtów zokrąglony do 1KiB to 1KiB, a ponieważ 1KiB nie jest mniejszy od 1Kib, to plik zostaje pominięty w wynikach (*sic! sic!*)

Oznacza to, że w pełni wiarygodne wyniki poszukiwania plików mniejszych niż pewna wartość są osiągalne tylko i wyłącznie wtedy, gdy użyjemy jednostki **c** (bajty), kiedy opisane wcześniej szaleńcze „zaokrąglanie” nie jest w stanie przynieść szkód. Sprawdźmy.

```
$ find -size -1024c
./0
./8
```

I dopiero teraz otrzymujemy wynik, którego oczekiwaliśmy.

Dodatkowo definiuje się pewną liczbę pseudo-warunków, których zadaniem nie jest badanie cech pliku, a wykonanie specyficznej akcji w chwili, gdy plik/katalog zostanie odnaleziony. Podajemy dwa z nich:

`-exec` *`polecenie`* `\;`
wykonuje dowolne *polecenie* w miejscu odnalezienia pliku; traktowany jest jako spełniony, gdy polecenie kończy się poprawnie (zwraca kod powrotu 0); w *poleceniu* można używać symbolu **`{}`**, oznaczającego nazwę odnalezionego pliku/katalogu

`-print`
traktowany tak, jakby zawsze był spełniony; powoduje wypisanie nazwy odnalezionego pliku do standardowego wyjścia; używany, gdy na skutek użycia innych warunków (na przykład **`-exec`**) domyślne wypisywanie nazw jest wyłączone

Powyższe warunki można dowolnie wiązać poniższymi operatorami (wymienionymi w kolejności malejących priorytetów):

`(` *`wyrażenie`* `)`
nawiasy – wymuszają zmianę wiązania operatorów w tradycyjny, matematyczny sposób; ponieważ powłoka może używać ich dla swoich celów, warto przygotować się na konieczność użycia cytowania **`\(`** i **`\)`**

`!` *`wyrażenie`*
negacja – tu również przygotuj się na użycie cytowania

`-not` *`wyrażenie`*
jak wyżej, ale wygodniej, bo bez cytowania

`wyrażenie1 wyrażenie2`
domyślne złączenie wyrażeń operatorem „i”

`wyrażenie1 -a wyrażenie2`
jawne złączenie wyrażeń operatorem „i” (**`-a`** jak *and*)

wyrażenie1* `-o` *wyrażenie2
jawne złączenie wyrażeń operatorem „lub” (`-o` jak *or*)

Przykłady użycia polecenie **`find`**:

`find ~-name abc.txt`
wyszukuje wszystkie pliki/katalogi o nazwie *abc.txt*, które mogą znajdować się w katalogu domowym użytkownika oraz w jego podkatalogach
`find ~/temp -name "*.txt"`
wyszukuje wszystkie pliki/katalogi o nazwie z przyrostkiem , które znajdują się w podkatalogu **`temp`** katalogu domowego użytkownika oraz ewentualnych podkatalogach tego katalogu
`find ~ -iname "*.txt" -type f -size +10M`
wyszukuje wszystkie pliki zwykłe w katalogu domowym użytkownika (i jego podkatalogach), których nazwy mają przyrostek **`.txt`** (wielkość liter bez znaczenia) oraz rozmiar większy niż 10 mebibajtów
`find /tmp -type f -atime +2 -exec rm {} \; -print`
wyszukuje w katalogu **`/tmp`** oraz jego podkatalogach wszystkie pliki zwykłe, których nie używano w żaden sposób w ciągu ostatnich 48 godzin; wszystkie odnalezione pliki zostaną usunięte; dodatkowy przełącznik **`-print`** spowoduje, że zostaną wyświetlone nazwy odnalezionych plików, pomimo wykonania na nich dodatkowej operacji (tutaj **`rm`**).

4.6. Źródła uzupełniające

1. Manual polecenia **`grep`**: `https://man7.org/linux/man-pages/man1/grep.1.html`
2. Opis implementacji wyrażeń regularnych w języku Perl: `https://perldoc.perl.org/perlre`
3. Opis implementacji wyrażeń regularnych w języku Python: `https://docs.python.org/3/howto/regex.html`
4. Opis implementacji wyrażeń regularnych w bibliotece Gnulib: `https://www.gnu.org/software/gnulib/manual/html_node/Regular-expressions.html`
5. Manual polecenia **`find`**: `https://man7.org/linux/man-pages/man1/find.1.html`

4.7. Zadania do samodzielnego wykonania

Wskazówka: do testowania swoich wyrażeń regularnych możesz wykorzystać taki oto prosty potok:

```
echo napis | grep -E 'wzorzec'
```

Dzięki niemu będziesz mógł na bieżąco obserwować zachowanie twoich wyrażeń i sposób działania polecenia **grep**.

1. W jaki sposób umieścić znaki **[** i **]** wewnątrz zbioru?
2. Czy twój system operacyjny jest zlokalizowany w sposób, który sprawia, że litera 'ą' jest elementem zbioru **[a-c]**?
3. Jakie znaki w napisie mogą dopasować się do klasy **[:cntrl:]**?
4. Korzystając z podręcznika systemowego zbadaj sposób użycia narzędzia **fgrep**.
5. Od jakiej wartości startują liczniki prezentowane przez przełączniki **-n** i **-b** polecenia **grep**?
6. Zbadaj działanie przełączników **-A** i **-B** polecenia **grep**.
7. Skonstruuj wyrażenia regularne sprawdzające, czy plik zawiera:
   - numery PESEL
   - polskie kody pocztowe
   - adresy IP
   - adresy e-mail
   - numer konta bankowego w standardzie IBAN ze spacjami rozdzielającymi lub bez nich
   - datę w formacie ISO 8601
   - puste linie
   - linie nie zawierające cyfr
   - linie potencjalnie zawierające imiona i nazwiska (dwa słowa obok siebie, rozdzielone białymi znakami, każde zaczynające się wielką literą)
   - znaki końca linii w standardzie systemu MS Windows
   - linie zaczynające się od liczb
   - linie nie kończące się liczbami
   - linie zawierające dokładnie jedną liczbę
8. Zbadaj działanie warunków **-maxdepth** i **-mindepth** polecenia **find**.
9. Skonstruuj linię poleceń uruchamiającą narzędzie **find** i odszukującą pliki o następujących cechach:
   - nieużywane od 2 lat

- mniejsze od 1KiB i większe od 100B
- większe od 1KiB lub mniejsze od 100B
- takie, których właścicielem jest ktoś inny niż ty
- wykonywalne w systemie MS Windows
- większe niż 100MiB i użyte w ciągu ostatniego miesiąca
- uruchamiane w ciągu ostatniego tygodnia

10. Powiąż ze sobą polecenia **`find`** i **`grep`** używając warunku **`-exec`** i symbolu **`{}`** w celu znalezienia plików:
    - których nazwy kończą się przyrostkiem **`text`** i które zawierają adresy email
    - które są mniejsze niż 10KB i które sprawiają wrażenie, że zawierają kod źródłowy w języku C/C++ (*wskazówka: co zwykle zawierają pierwsze linie pliku źródłowego w tych językach?*)