Wbrew pozorom w pracy administratora systemów nader często pojawia się konieczność wyszukania informacji w dzienniku systemowym, modyfikacji plików konfiguracyjnych albo utworzenie statystyk. W tych, ale i w wielu innych codziennych zadaniach związanych z zarządzaniem systemami, niezbędne są narzędzia do przeszukiwania i automatycznej edycji plików tekstowych.

Jak wiemy, do wyszukiwania plików spełniających określone kryteria służy niezbędnik administratora `find`. Z kolei do przeszukiwania zawartości plików tekstowych niezastąpione są programy:

- `fgrep`,
- `grep`,
- `egrep`.

Nazywają się podobnie i działają podobnie. Różnica polega na tym jak wykorzystują wyrażenia regularne. Pierwszy z nich, `fgrep` w ogóle nie wykorzystuje wyrażeń regularnych. Wyszukuje wyłącznie literalnie podany ciąg znaków. Ponieważ kompilacja wyrażeń regularnych jest czasochłonna i zasobochłonna, w ten sposób `fgrep` jest najszybszy z całej trójki. Stąd jego nazwa *fast grep*.

W praktyce, w 75 przypadkach na 100, administratorowi wystarczy wyszukiwanie bez wyrażeń regularnych. Na przykład, aby zobaczyć jakie moduły jądra programu VirtualBox są załadowane w danym momencie, wystarczy wpisać:

```
sirius$ modinfo | fgrep vbox
206 ffffffffeffe3000  51740 232   1  vboxdrv (VirtualBox HostDrv 5.0.20)
207 ffffffffef98d5d8    cb0 233   1  vboxnet (VirtualBox NetAdp 5.0.20)
209 ffffffff0028000   7320 234   1  vboxflt (VirtualBox NetDrv 5.0.20)
209 ffffffff0028000   7320   -   1  vboxflt (VirtualBox NetMod 5.0.20)
```

i błyskawicznie mamy żądaną listę 4 modułów spośród wszystkich 250. Tak jak w tym przykładzie, bardzo często używa się `fgrep` jako filtra w potoku z innymi programami. Wówczas przeszukiwane jest standardowe wejście. Tutaj `modinfo` wypisze wszystkie moduły załadowane do jądra systemu, natomiast z tej listy `fgrep` odfiltruje tylko te linie tekstu, które zawierają ciąg znaków `vbox`.

Gdy znamy identyfikator wiadomości poczty elektronicznej możemy łatwo odszukać w dzienniku `/var/log/syslog` wszystkie wpisy systemu pocztowego na temat tej wiadomości:

```
math$ fgrep 10HKRPXu028911 /var/log/syslog
Jan 17 21:27:26 math sendmail[28911]: [ID 801593 mail.info] 10HKRPXu028911:
from=<fail2ban@math.uwb.edu.pl>, size=433, class=0, nrcpts=1,
msgid=<202101172027.10HKRPQ9028907@math.uwb.edu.pl>, proto=ESMTPS,
daemon=MTA, relay=localhost [127.0.0.1]
Jan 17 21:27:26 math sendmail[28907]: [ID 801593 mail.info] 10HKRPQ9028907:
to=mariusz@math.uwb.edu.pl, delay=00:00:01, xdelay=00:00:01, mailer=relay,
pri=30198, relay=[127.0.0.1] [127.0.0.1], dsn=2.0.0, stat=Sent
(10HKRPXu028911 Message accepted for delivery)
Jan 17 21:27:26 math sendmail[28913]: [ID 801593 mail.info] 10HKRPXu028911:
to=<mariusz@math.uwb.edu.pl>, delay=00:00:00, xdelay=00:00:00, mailer=cyrus,
pri=120433, relay=localhost, dsn=2.0.0, stat=Sent
```

albo po adresie e-mail możemy szybko odnaleźć kiedy i skąd dostarczona została poczta na dany adres:

```
math$ fgrep mf.gov.pl /var/log/syslog*
```

Zauważmy w tym przykładzie, że przeszukiwany nie jest jeden plik, ale kilka. Aktualnie używany to `/var/log/syslog`, natomiast starsze pliki tego dziennika otrzymują co jakiś czas kolejny numer od 0 do 7. Im większy numer tym starszy plik. Dodając gwiazdkę fgrep dostaje całą listę plików do

przeszukania. Domyślnie poprzedzi każdą linię odpowiedzi nazwą pliku, z którego ta linia tekstu pochodzi. Możemy to wyłączyć podając opcję `-h`.

Czasem interesuje nas wyłącznie ile razy dany ciąg znaków występuje w pliku. Na przykład:

```
sirius$ modinfo | fgrep -c vbox
4
```

Nader często będzie interesować nas tylko to, gdzie dany ciąg znaków wystąpił, nie ilość wystąpień, ani też tekst, w którym ten ciąg się znajduje. Dobry przykład to, gdy mamy kilkadziesiąt plików kodu źródłowego a chcemy odnaleźć te pliki, które zawierają daną frazę. Na przykład:

```
sirius$ find . -type f -name "*.php" -exec fgrep -l getPracownik {} \;
./lib/layout.php
./en/erasmus.php
./en/badania/seminarium.php
./templates/jednostki/jednostka.php
./templates/wladze.php
./templates/zaklady/zaklad.php
./pl/studia/erasmus.php
./pl/studia/opiekunowie.php
./pl/studia/praktyki_zawodowe.php
./pl/studia/kola_naukowe.php
./pl/badania/seminarium.php
./pl/ckum/kontakt.php
./pl/ckum/index.php
./pl/systemjk/index.php
```

Mamy tutaj przykład jednoczesnego użycia programów `find` i `fgrep` przez exec, nie w potoku. Program `find` wyszukuje w podanym miejscu pliki o rozszerzeniu `php`, a `fgrep` w każdym z nich szuka frazy `getPracownik` (akurat tutaj to nazwa funkcji w kodzie PHP). Ponieważ jest opcja `-l` to `fgrep` zwraca tylko nazwę pliku, gdy znajdzie w nim szukaną frazę. Przy okazji, to samo można uzyskać stosując shellowe command substitution:

```
sirius$ fgrep -l getPracownik `find . -type f -name "*.php"`
./lib/layout.php
./en/erasmus.php
./en/badania/seminarium.php
./templates/jednostki/jednostka.php
./templates/wladze.php
./templates/zaklady/zaklad.php
./pl/studia/erasmus.php
./pl/studia/opiekunowie.php
./pl/studia/praktyki_zawodowe.php
./pl/studia/kola_naukowe.php
./pl/badania/seminarium.php
./pl/ckum/kontakt.php
./pl/ckum/index.php
./pl/systemjk/index.php
```

Tutaj najpierw program `find` buduje listę plików o rozszerzeniu `php`, która trafia do `fgrep` jako argument. Wynik jest taki sam. Co jest lepsze? Kwestia gustu chyba.

Gdybyśmy chcieli zobaczyć numery linii, w których występuje funkcja `getPracownik()` to użyjemy opcji `-n`:

```
sirius$ fgrep -n getPracownik ./templates/wladze.php
10:$page->pushHeader($page->getPracownik(183), 2);
23:$page->pushHeader($page->getPracownik(192), 2);
36:$page->pushHeader($page->getPracownik(133), 2);
```

Podając wzorzec do wyszukiwania należy pamiętać o znakach specjalnych shella `*`, `?`, `[`, `]`, `$`, `^`, `|`, `(`, `)`, `\` i spacji. Wprawdzie `fgrep` traktuje je literalnie, ale shell może je interpretować, dlatego należy je chronić. Tak więc, na przykład:

```
fgrep ”ala ma kota” /tmp/plik.txt
```

Do ochrony spacji wystarczą cudzysłowy, ale pozostałe znaki specjalne bezpieczniej jest chronić apostrofami:

```
fgrep '$zmienna' /tmp/plik.txt
```

Powstaje pytanie jak ochronić cudzysłów albo apostrof? Cudzysłowy ochronimy apostrofami, ale zawsze można użyć znaku backslash `\`, który chroni wszystko.

Program `grep` różni się od `fgrep` praktycznie tylko tym, że pozwala używać wyrażeń regularnych w wersji podstawowej, ale w wielu wypadkach to wystarczy. Opcje i sposób działania są niema identyczne. W praktyce może rzadziej używa się wyrażeń regularnych podczas wyszukiwania, ale są one niezastąpione. Gdy chcemy obejrzeć zawartość pliku konfiguracyjnego bez komentarzy, które oznaczone są znakiem `#` na początku linii, jak na przykład w pliku `/etc/nsswitch.conf`, którego początek wygląda tak:

```
math$ head -20 /etc/nsswitch.conf
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# /etc/nsswitch.dns:
#
# An example file that could be copied over to /etc/nsswitch.conf; it uses
# DNS for hosts lookups, otherwise it does not use any other naming service.
#
# "hosts:" and "services:" in this file are used only if the
# /etc/netconfig file has a "-" for nametoaddr_libs of "inet" transports.

# DNS service expects that an instance of svc:/network/dns/client be
# enabled and online.

passwd:     files
group:      files
```

wystarczy wpisać:

```
math$ grep -v ^# /etc/nsswitch.conf



passwd:     files
```

```
group:      files
```

Podany wzorzec `^#` pasuje do znaku `#` na początku wiersza. Opcja `-v` spowoduje wypisanie wszystkich linie, które nie pasują do podanego wzorca. W wyniku mogą pojawić się puste linie, których nie chcemy. Aby je wyeliminować można dodać dodatkowy filtr używając `grep` i wyrażeń regularnych:

```
math$ grep -v ^# /etc/nsswitch.conf | grep .
passwd:     files
group:      files
hosts:      files dns
ipnodes:    files dns
networks:   files
```

Wyrażenie `.` (pojedyncza kropka) będzie dopasowane do dowolnego znaku. W ten sposób pominięte zostaną puste line, bo nie zawierają żadnych znaków. Bez wyrażeń regularnych nie uzyskalibyśmy tego, a już na pewno nie tak łatwo.

Gdybyśmy chcieli wyszukać jakie adresy poczty e-mail zostały użyte w plikach źródłowych PHP strony internetowej jaką opracowujemy to można połączyć `grep` z `find`:

```
find . -name "*.php" -exec grep "[a-z-_.]*@[a-z-_.]*\.[a-z]*" {} \;
```

Program `find` odszukuje pliki, które kończą się na `.php` i dla każdego z nich uruchamia `grep`. Grep szuka linii kodu zawierających ciąg pasujący do podanego wyrażenia regularnego. Ponieważ wyrażenie to zawiera znaki specjalnie traktowane przez shella `[,]`, `*`, `\`, więc należy je ochronić. Dlatego wyrażenie ujęte jest w cudzysłowy.

Opanowanie programu `grep` tak na prawdę sprowadza się do opanowania wyrażeń regularnych, bo samo narzędzie jest bardzo proste i używa się go niemal tak samo jak `fgrep`. Oczywiście ciąg znaków nie zawierający znaków specjalnych stosowanych w wyrażeniach regularnych mozna traktować jako bardzo proste wyrażenie regularne. Dlatego możemy używać `grep` jak `fgrep`, na przykład:

```
grep root /etc/passwd
fgrep root /etc/passwd
```

Otrzymamy te same wyniki, ale `fgrep` zrobi to samo szybciej. Teraz komputery są bardzo szybkie i pewnie nikt nie zauważy różnicy w czasie działania obu poleceń. Gdy jednak używamy `grep` w połączeniu z `find` dla wielu plików, to czas działania może już trochę się różnić.

Jeśli chcemy używać pełnych wyrażeń regularnych, które zawierają znaki specjalne `+`, `?`, `|`, `{`, `}`, to potrzebujemy bardziej zaawansowany algorytm w jaki wyposażony jest program `egrep`. Jego nazwa pochodzi od *expression grep* albo od *extended grep*. Z uwagi na rozbudowany algorytm jakiego używa `egrep` jest wolniejszy od `grep`, dlatego używa się go raczej rzadko.

Szukając adresów e-mail użytych w kodzie PHP możemy użyć rozbudowanych wyrażeń regularnych:

```
find . -name "*.php" -exec egrep "[a-z-_.]{3,}@[a-z-_.]{2,}\.[a-z]{2,}" {} \;
```

Zestaw opcji `egrep` jest niemal identyczny jak dla `fgrep` i `grep`. Ciekawa jest opcja `-o` pozwalająca wypisać wyłącznie dopasowaną frazę do wyrażenia regularnego.