Polecenia systemu plików

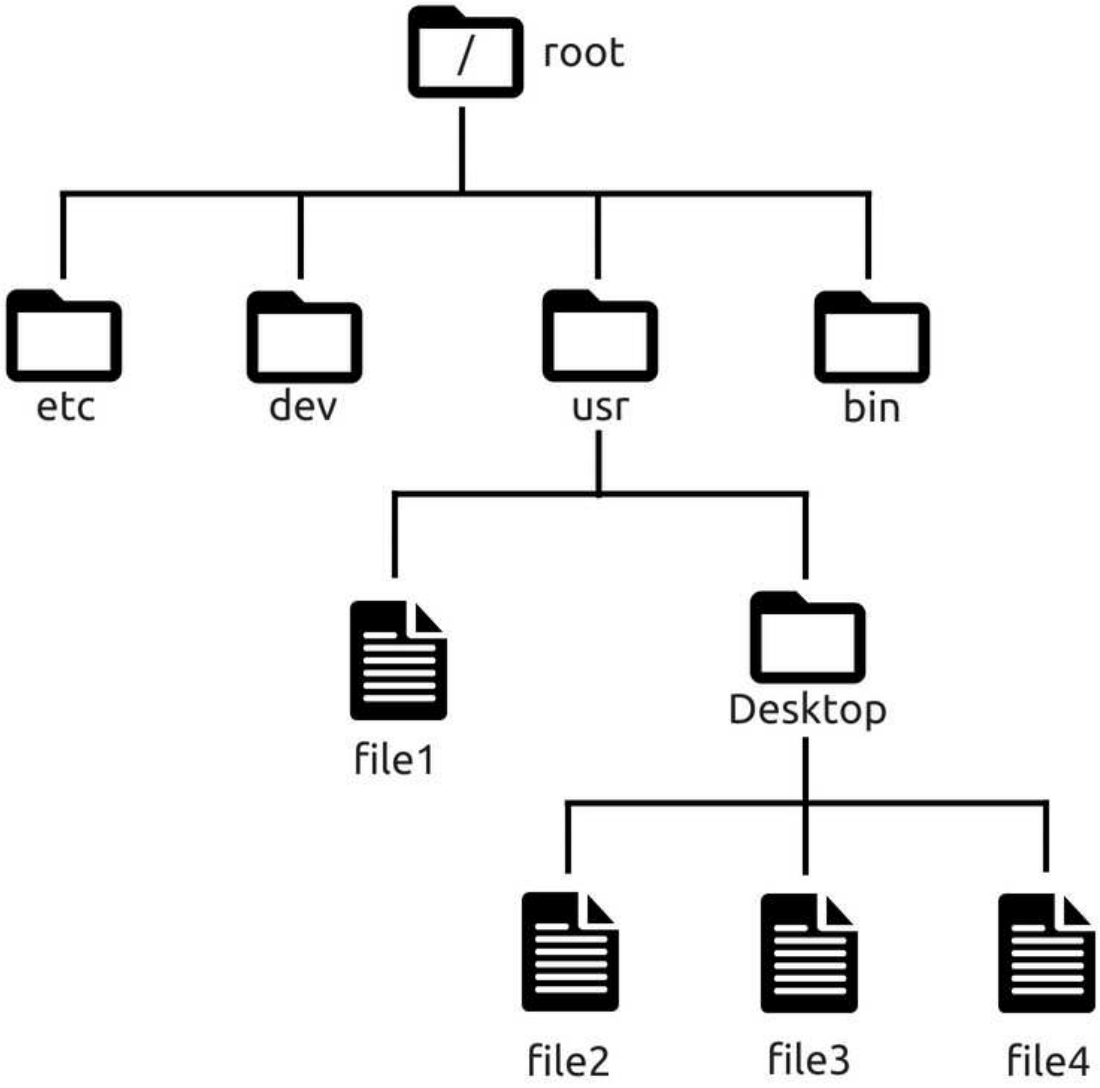
System plików to miejsce trwałego przechowywania danych w systemie UNIX. Każda bezpośrednia ścieżka do pliku wygląda podobnie:

```
/user/katalog/podkatalog/plik.rozszerzenie
```

np.

```
/rita/Desktop/Pictures/sea.png
```

Intucyjnie jest to dla nas zrozumiałe. Na moim pulpicie znajduje się katalog Pictures zawierający obrazek sea.png.
Jednak zapis ten nie jest przypadkowy. Wynika z faktu, że **system plików ma strukturę hierarchiczną**, a początkiem jest zawsze katalog główny zwany `root` i oznaczany symbolem: `/`.



1. `pwd` – pokaż ścieżkę do katalogu bieżącego

Polecenie `pwd` (*Print Working Directory*) wyświetla aktualną ścieżkę do katalogu – folderu, w którym własnie jesteśmy.

```
$ pwd
~/rita/
```

Linux pwd: zawsze zgubionym odpowie na pytanie „gdzie jestem? jaka jest moja ścieżka?”

2. `ls` – wylistuj zawartość katalogu

Jedna z najczęściej używanych komend systemu Linux, `ls` (*List*) wyświetla całą zawartość folderu (pliki i foldery), w którym się znajdujemy.

```
$ ls
Desktop  Documents  Downloads  Pictures  Videos
```

Linux ls: jedna z najczęściej używanych komend – jeszcze zobaczysz w tym tutoralu 😉

Wzorce uogólniające

W folderze, który zawiera wiele plików i podfolderów może być trudno nam znaleźć intresujące informacje. Wówczas przydadzą się wzorce uogólniające np. `*, ?, []`

`*` – zastępuje dowolny ciąg znaków
`?` – oznacza dowolny jeden znak
`[]` – zastępuje jeden znak, ale sposród znaków podanych w nawiasie

Załóżmy, że przeszłam do folderu Pictures, gdzie mam bardzo wiele obrazków zapisanych pod różnymi nazwami np. *image01, image1, image2, img1, img2, img3, picture1, picture2 itp.*

Załóżmy, że chcę znaleźć tylko te pliki, których nazwa zaczyna się na img. Stąd połączę *img* z gwiazdką:

```
$ ls img*
img1  img2  img3
```

Mogę też wyszukać obrazki, które zaczynają się dwoma literami *im* a kończą cyfrą *1*:

```
$ ls im*1
image01  image1  img1
```

albo wszystkie, które są zakończone cyfrą *1*(czyli zaczynają się dowolnym ciągiem znaków i występuje 1 na końcu:

```
$ ls *1
image01  image1  img1  picture1
```

Możemy też wyszukać takie pliki, w których nazwa to *image*, a kończą się dowolnym, jednym znakiem:

```
$ ls image?
image1  image2
```

To samo uzyskamy zadając zbiór cyfr w nawiasie – [12]:

```
$ ls image[12]
image1  image2
```

Wzorców uogólniających jest znacznie więcej, ale te trzy podstawowe wystarczą nam na start.

Pliki i katalogi ukryte

Bywa tak, że folder zawiera pliki i podkatalogi, które w trybie wizualnym są ukryte.

Nazwy plików ukrytych często zaczynają się od kropki. Jeśli używamy wzorca `*` to taka ukryta zawartość się nie pojawi, ale uzywając `ls .*` możemy wyszukać pliki ukryte. Co jeśli jednak ukryty folder czy plik ma zupełnie inną nazwę? Wtedy warto użyć komendy `ls` z **przełącznikiem** np. z opcją -a.

Komendy:
`ls -a`
`ls -la`

Wyświetlają także ukryte pliki i katalogi.

3. `cd` – zmiana katalogu

Za pomocą `cd` (*Change Directory*) zmieniamy miejsce / katalog w którym się znajdujemy.

Z katalogu *rita* chcę przejść na pulpit:

```
$ cd Desktop
$ pwd
~/rita/Desktop
```

Linux cd: poruszanie się po katalogach

```
$ cd ..
$ pwd
~/rita/
```

Za pomocą `cd ..` cofniemy się do katalogu poprzedniego.

Ponadto możemy podać całą ścieżkę do folderu, do którego chcemy wskoczyć. Nie musimy nawet pamiętać pełnych nazw. Wystarczy, że wpiszemy 2-3 pierwsze litery z nazwy i klikniemy klawisz Tab, a terminal sam podpowie jakie mamy dostępne foldery w tej lokalizacji.

Np. na pulpcie mam podfolder From

```
$ cd Desktop/Form
$ pwd
~/rita/Desktop/Form
```

```
$ cd ..
$ pwd
~/rita/Desktop
```

4. `man` – pokaż pomoc

Poleceniem `man` (*manual*) wyświetlimy pomoc dla danej komendy i czasem przykład użycia. Np. `man cd` wyświetli opis – jak używać komendy cd, jaki ma opcje. Użyj `q`, żeby wyjść z trybu czytania pomocy.

```
$ man ls
LS(1)                           User Commands                           LS(1)

NAME
       ls - list directory contents
DESCRIPTION
       List  information  about  the FILEs (the current directory by default).

...
```

Linux man: wyświetli pełny opis użycia komendy (tutaj ucięte)

5. `mkdir` – tworzenie katalogu

Czas stworzyć nowy folder z komendą `mkdir`(*Make Directory*).

```
$ mkdir new_folder
$ ls
Desktop  Documents  Downloads  new_folder  Pictures  Videos
```

Linux mkdir: tworzenie katalogu linux'owego jest banalnie proste

6. `rmdir` – usuwanie katalogu

Usuwa wybrany folder z komendą `rmdir`(*Remove Directory*).

```
$ rmdir new_folder
$ ls
Desktop  Documents  Downloads  Pictures  Videos
```

Linux rmdir: usuwanie jest odwrotnością tworzenia

Za pomocą `rmdir` można usunąć wiele katalogów, podając je jeden po drugim oddzielone spacją.

Jeżeli usuwany katalog jest niepusty konsola może zgłaszać błąd. Jeżeli jesteśmy absolutnie pewni, że wiemy, które katalogi usuwamy (niezależnie od zawartości) to możemy użyć opcji `--ignore-fail-on-non-empty`, która wyciszy komunikat o folderach zawierających pliki i usunie je w całości.

7. `touch` – tworzenie pliku

Po folderach czas na utworzenie nowego pustego pliku.

```
$ touch new_file.txt
$ ls
Desktop  Documents  Downloads  new_file.txt  Pictures  Videos
```

Linux touch: tworzenie pliku jeśli plik nie istnieje

Touch aktualizuje datę ostatniego otwarcia pliku.

8. `cat` – łączenie plików

Innym sposobem utworzenia pliku może być wykorzystanie polecenia `cat`(*Concatenate*). Komenda `cat` służy przede wszystkim do konkatenacji (łączenia) plików, ale może być również wykorzystana do wyświetlenia plików czy stworzenia nowego pliku.

Najczęściej programu `cat` używa się w ten sposób:

```
$ cat plik1.txt plik2.txt
```

Powyższy przykład wyświetli na ekranie połączone dwa pliki.
Możemy wynik tego działania zapisać do nowego pliku:

```
$ cat plik1.txt plik2.txt > plik3.txt
```

Linux cat: konkatencja i zapis do pliku

Komenda `$ cat > sample.txt` pozwala na bezpośrednie wypełnienie pliku z konsoli. Możemy napisać dowolny ciąg znaków, a następnie zapisać zmiany i wyjść za pomocą (w zależności od systemu) Ctrl+D / Ctrl+C / Cmd + C.

```
$ cat > sample1.txt
...
$ ls
Desktop  Documents  Downloads  new_file.txt  Pictures  sample1.txt  Videos
```

Znak `>` oznacza *przeadresowanie standardowego wyjścia*. Prościej mówiąc, jeśli naszym **urządzeniem standardowego wyjścia jest monitor**. Napisane zdanie zostałoby tylko wypisane na ekranie. Przekierwując wyjście do pliku – tworzymy nowy plik i zapisujemy w nim to, co byłoby tylko wyświetlone przez cat.

Dlatego też, jeżeli chcemy utowrzyć czysty plik, możemu zupełnie pominąć polecenie `cat` i utworzyć nowy, pusty plik w ten sposób:

```
$ > sample2.txt
$ ls
Desktop  Documents  Downloads  new_file.txt  Pictures  sample1.txt  sample2.txt  Videos
```

9. `cp` – skopiuj

Do kopiowania plików słóży polecenie `cp`(*Copy*). Możemy utworzyć kopię pliku w tym samym folderze pod nową nazwą

```
$ ls
Desktop  Documents  Downloads Pictures  sample.txt  Videos
$ cp sample.txt new_sample.txt
$ ls
Desktop  Documents  Downloads  new_sample.txt  Pictures  sample.txt  Videos
```

Linux cp: kopiowanie przyjmuje 2 parametry: nazwę lub ścieżkę kopiowanego pliku/katalogu oraz nazwę lub ścieżkę miejsca, do którego kopiujemy

Możemy też utworzyć kopię pliku pod tą samą nazwą, ale w podkatalogu:

```
$ cp sample.txt Pictures
```

Polecenie skopiuje plik sample do folderu Pictures pod nazwą sample.txt

Możemy też uworzyć kopię pliku, pod nową nazwą, ale w podkatalogu:

```
$ cp sample.txt Pictures/new_sample.txt
```

Polecenie skopiuje plik sample do folderu Pictures pod nazwą new_sample.txt

Linux `cp` z opcją `-r`pozwala również kopiować całe foldery:

```
$ cp -r Pictures New_Pictures
```

10. `mv` – przenieś

Polecenie `mv` (*Move*) służy to przeniesienia pliku do innego folderu.

```
$ mv sample.txt Documents
```

Linux mv: przenoszenie i lub zmiana nazwy

Można za jego pomocą również zmienić nazwę pliku:

```
$ mv sample.txt new_file.txt
```

11. `rm` – usuń

Kiedy kończymy pracę z plikiem i chcemy go usunąć wystarczy użyć komendy `rm`(*Remove*):

```
$ rm sample.txt
```

Linux rm: usuwanie plików i folderów

Z opcją `-r` usuniemy także katalogi – tak samo jak `rmdir`.

Procesy

Każdy program, który działa na komputerze nazywany jest **procesem**. Na pewno macie programy, które startują wraz z uruchomieniem komputera – antywirus, program do

aktualizacji, skype czy slack – są to przykłady procesów uruchamianych ze startem, ale jest też sporo, o których nie macie pojęcia a są konieczne do działania systemu.

Każdy proces w systemie ma swój identyfikator, za pomocą, którego można go jednoznacznie wskazać, taki numer nazywa się PID – *process IDentifier*.

Spotkaliście się kiedyś z sytuacją, że uruchominy program nie dało się w żaden sposób zamknąć – nie reagował na klikanie, prawy przycisk myszy czy skróty klawiaturowe? Za pomocą PID można proces znaleźć i skutecznie zabić – zakończyć jego pracę.

12. `ps` – pokaż procesy

Polecenie `ps` (*Process Status*) wyświetla listę procesów dla aktualnej powłoki:

```
$ ps
  PID TTY          TIME CMD
    4 tty1     00:00:00 bash
   45 tty1     00:00:00 ps
```

Linux ps: lista procesów działających w bieżącej powłoce

`ps`– wyświetliło tylko podstawowe informacje o prcesach powłoki – pid, terminal, czas aktywności oraz nazwę procesu. Możemy otrzymać pełne informacje o procesie aktualnej powłoki za pomocą `ps -f` (*full*):

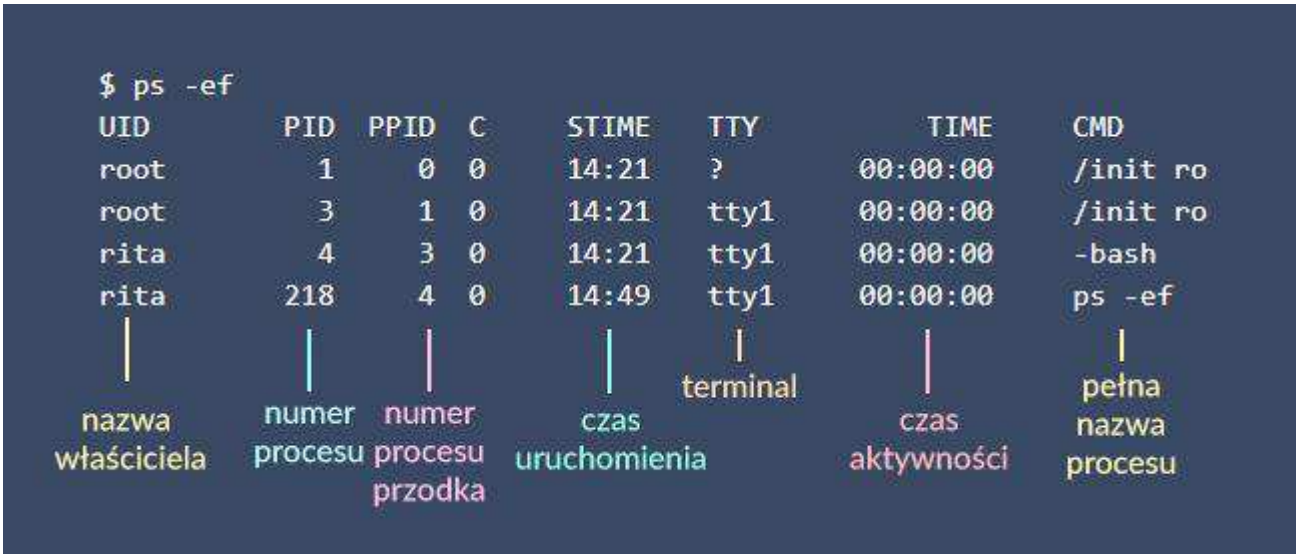
```
$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
rita         4     3  0 14:21 tty1     00:00:00 -bash
rita       116     4  0 14:38 tty1     00:00:00 ps -f
```

Aby wyświetlić wszystkie procesy możemy użyć polecenia `ps -e` (*every*) lub `ps -A` (*all*) – można spotkać oba użycia, zwracają to samo.

```
$ ps -e
  PID TTY          TIME CMD
    1 ?        00:00:00 init
    3 tty1     00:00:00 init
    4 tty1     00:00:00 bash
  202 tty1     00:00:00 /Applications/GoogleChrome.app
  228 tty1     00:00:00 ps
```

Możemy łączyć opcje np `-ef`:

```
$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 14:21 ?        00:00:00 /init ro
root         3     1  0 14:21 tty1     00:00:00 /init ro
rita         4     3  0 14:21 tty1     00:00:00 -bash
rita       202     3  0 14:25 tty1     00:00:00 /Applications/GoogleChrome.app
rita       218     4  0 14:49 tty1     00:00:00 ps -ef
```



13. `kill` – zakończ proces

Dowolny proces możemy zakończyć za pomocą polecenia `kill` podając jako argument PID procesu, który chcemy usnuąć

```
$ kill 202
```

Tym poleceniem wysłałam sygnał – tutaj zakończenia do procesu 317 – u mnie przeglądarki. Może się jednak zdarzyć, że proces nie zostanie zakończony, gdyż czeka na inny proces np. zakończenie zapisu, lub po prostu mamy problem z „zawieszonym” programem i chcemy go raz, a skutecznie zakończyć.

W takim wypadku możemy przesłać opcję `-9` – sygnał bezwarunkowego zakończenia procesu.

```
$ kill -9 202
```

Linux kill: użycie polecenia z przełącznikiem -9 spowoduje natychmiastowe zamknięcie, ale bez zapisania pracy!

Narzędzia pomocnicze

Unix udostępnia nam wiele komend i programów pomocniczych, które przyspieszają pracę – przeszukiwanie katalogów czy dostęp do plików.

14. `locate` – zlokalizuj

Metoda `locate`, która często jest najprostrzą i najszybszą metodą do znalezienia lokalizacji pliku lub folderu o podanej nazwie. Wielkość liter ma znaczenie!

```
$ locate Downloads
~/rita/Desktop/Downloads
```

możemy też zmusić locate do policzenia (*count*) liczby trafień:

```
$ locate -c Downloads
1
```

Niestety metoda ta, chociaż jest bardzo szybka, ma też wadę – **nie odnosi się bezpośrednio do drzewa systemu plików**, a korzysta z pewnego rodzaju referencji (bazy) plików. Może się, więc zdarzyć, że metoda znajdzie plik, który został już usunięty (a jego referencja jeszcze nie) lub nie znajdzie pliku, który jeszcze nie pojawił się w bazie (wymaga uruchomienia `sudo updatedb`).

15. `find` – znajdź

Bardziej popularna metoda do wyszukiwania plików i folderów to `find`. Przeszukuje drzewo katalogów, aby znaleźć pliki i foldery spełniające warunek.

Wylistuj pliki w folderze i pofoldery

Bez parametrów polecenie wyświetli listę wszystkich plików w bieżącym katalogu, a także podkatalogi w bieżącym katalogu.

```
$ find
.
./aaa.txt
./pod_folder
./pod_folder/abc.rb
./pod_folder/test.rb
./wow.html
./wow.rb
```

`.` – kropka na początku ścieżki oznacza „w bieżącym folderze – folderze, w którym jestem”

Szukanie danego folderu lub ścieżki

Poniższe polecenie wyszuka wszystkie pliki w katalogu pod_folder w bieżącym katalogu.

```
$ find ./pod_folder
./pod_folder
./pod_folder/abc.rb
./pod_folder/test.rb
```

Wyszukanie za pomocą wzorca

Używając przełącznika `-name` możemy wyszukiwać pliki po nazwie. Pamiętacie, że wzorzec `*` – oznacza wybierz wszystko?

Możemy te informacje połączyć i wyszukać wszystkie pliki ruby znajdujące się w naszym folderze i jego podfolderach:

```
$ find -name '*.rb'
./pod_folder/abc.rb
```

```
./pod_folder/test.rb
./wow.rb
```

Wyszukanie przez zaprzeczenie

Może się zdarzyć tak, że chcemy wyszukać wszystkie pliki, które nie spełniają pewnego warunku np. nie mają w nazwie jakiegoś słowa, albo chcemy odsiać pliki o pewnym rozszerzeniu np. w folderze obrazy wyświetlić wszystkie pliki, które nie mają rozszerzenia .jpg, gdyż prawdopodobnie zostały tam zapisane przypadkiem. Tu z pomocą przyjdzie opcja `not`.

Poniższy przykład wyświetli wszystkie pliki i podfoldery w katalogu, które nie są plikami ruby:

```
$ find not -name '*.rb'
./aaa.txt
./pod_folder
./wow.html
```

Wyszukiwanie tylko plików lub folderów

Czasami możemy chcieć znaleźć tylko pliki to zadanej nazwie lub tylko foldery. Do tego służy przełącznik `-type` (f – file / d -directory).

W naszym folderze znajduje się zarówno plik xyz, jak i podfolder o takiej samej nazwie:

```
$ find -name 'xyz*'
./xyz.txt
./xyz
```

```
$ find -type f -name 'xyz*'
./xyz.txt
```

```
$ find -type d -name 'xyz*'
./xyz
```

16. `grep` – dopasuj do wzoru

Program `grep` (*Global Regular Expression Print*) – służył początkowo wyszukiwaniu wzorców w pliku i był podstawowym programem systemu UNIX.
Polecenie `grep` przegląda aktualny katalog w poszukiwaniu plików spełniających warunek – zawierających wzorzec w zawartości i pasujące do nazwy pliku

`grep 'wzorzec_tekstu' nazwa_pliku`
np.

```
$ grep 'hello' sample.txt
```

– przeszuka plik sample.txt czy zawiera słowo hello.

```
$ grep 'hello' sum*
```

Powyższe polecenie przeszuka wszystkie pliki, których nazwy zaczynają się na „sum" (gwiazdka dopasowuje dowolny ciąg znaków) – np. *summary.txt, summer.pdf, summit.doc* i wyświetli na ekranie tylko te linie, które zawierają tekst „hello". Jeżeli plik summary.txt zawierałby zdanie „hello world!" to zostałoby ono wyświetlone na ekranie.

Wzorzec (u nas ‚hello') jest podawany jako wyrażenie regularne, które pozwala na dopasowanie jednego lub więcej wyrazów / ciągów znaków, służą do tego znaki lub wyrażenia specjalne np.

- `.` – kropka zastępuje dowolny 1 znak tekstu
- `[abc]` – oznacza wstawienie dowolnego znaku spośród podanych – a,b, lub c
- `[a-z]` – oznacza dowolny znak z zakresu a do z
- `[^]` – zaprzeczenie np. [^aA] – oznacza dowolny znak nie będący a lub A, natomiast [^a-z] oznacza dowolny znak różny od a do z
- `^` – niestety tutaj mała zmyłka – sam znak ^ oznacza początek linii
- `$` – oznacza koniec linii

Wszystko wygląda w porządku, ale co jeśli chcemy wyszukać tekst na podstawie znaku, który jest znakiem specjalnym?
Wystarczy ten znak poprzedzić symbolem backslash `\` np. * oznacza gwiazdkę, a nie wzorzec: dopasuj wszystko.

Opcje – polecenie grep

Grep przyjmuje też wiele przydatnych opcji np:
`-i` – wzorzec **NIE**uwzględnia wielkość liter
`-n` – dla każdego pliku zostanie też wyświetlony numer linii, w której został znaleziony wzorzec
`-l` – wyświeli tylko nazwy plików, gdzie został znaleziony wzorzec
`-v` – zaprzeczenie wzorca – wyświetli wszystkie linie, które **NIE** zawierają wzorca
`-w` – wyszukuje nie fragment tekstu, a cały wyraz pasujący do wzorca

Przykłady:
`$ grep '^[0-9]' sam*` – szuka w plikach sam* linii zaczynających się od cyfry
`$ grep '[eE]$' sam*` – szuka w plikach sam* linii kończących się na e lub E
`$ grep '\$' Pictures/*` – szuka w plikach podkatalogu Pictures linii zawierających znak dolara
`$ grep -nwv 'hello' sam*` – wyświetli numer linii plików sam*, które nie zawierają w środku wyrazu ‚hello'

17. `head` i `tail` – pokaż linie od początku / końca

Dwa programy pozwalające wyświetlić tylko poczatek (`head`) lub koniec *`tail`) pliku. Domyślnie wyświetlają po 10 pierwszych lub ostatnich linii tekstu. Za pomocą opcji `-n`– gdzie za n podstawiamy dowolną liczbę, możemy dowolnie zmienić wartość wyświetlanych linii.

W pliku pan_tadeusz.txt – mamy zapisany pierwszy rozdział Pana Tadeusza możemy łatwo wyświetlic pierwsze 20:

```
$ head -20 pan_tadeusz.txt
```

i ostatnie 15 linii tekstu:

```
$ tail -15 pan_tadeusz.txt
```

Można też skorzystać z wzorców uogólniających:

```
$ head -12 tekst[12].txt
```

Powyższe polecenie wyświetli z plików o nazwie tekst1.txt lub tekst2.txt pierwsze 12 linii.

18. `clear` – wyczyść terminal

Kiedy na naszym ekranie zrobił się już niezły bałagan, można szybko wyczyścić okienko terminala wpisując polecenie `clear`.

19. `history` – pokaż historię komend

Aby cofnąć się do poprzedniej komendy wystarczy nacisnąć strzałkę w górę w terminalu, możeny tak cofnać się kilka, kilkanaście poleceń. Jeśli jednak, chcemy sprawdzić historię wpisywanych komend przyda nam się polecenie `history`, które wypisze na ekranie historię użytych komend.

20. `exit` – zamknij terminal

Komenda `exit` zamyka terminal lub otwartą kartę.

Kilka słów o standardowym wejściu / wyjściu

Każdy proces domyślnie korzysta ze standrdowego wejścia/wyjścia – wejściem jest klawiatura terminala, czyli to czym wprowadzamy komendy i dane, natomiast wyjściem – ekran terminala, na którym dane są wyświetlane.

Przeadresowanie standardowego wejścia następuje przez przekierowanie wyniku wykonanego polecenia do pliku zamiast na ekran komputera.

`polecenie < plik` – przeadresowanie standardowego wejścia = pobranie danych wejściowych z innego źródła niż klawiatura np. pobranie z parametrów z pliku dla komendy

`polecenie > plik` – przeadresowanie standardowego wyjścia – wynik polecenia zostanie wysłany do pliku (zostanie utworzony nowy lub istniejący plik o tej nazwie zostanie nadpisany)

jeżeli nie chcemy stracić zawartości pliku to warto skorzystać z
`polecenie >> plik` – przeadresowanie standardowego wejścia, wynik polecenia zostaje wysłany do pliku i dopisany na końcu

```
$ history > history.txt
```

Powyższe polecenie zapisze nam historię poleceń do pliku.

Wyświetl 5 pierwszych poleceń z pliku history.txt i przeadresuj do nowego pliku np. history5.txt.

Potoki

Przeadresowanie wejścia / wyjścia możemy wykorzystać do tworzenia potoków. W ten sposób, aby otrzymać interesujący nas wynik angażujemy kilka róznych procesów. Każdy kolejny proces w potoku bierze dane wejściowe z poprzedniego procesu (przeadresowane wyjście).

Zamiast wyjście polecenia `history`przeadresować do pliku użyjmy potoku do wyświetlenia pierwszych 5 komend:

```
$ history | head -5
```

Możemy też znaleźć wszystkie użyte do tej pory polecenia zawierające `ls`:

```
$ history | grep ls
```

i ostatecznie wyświetlić tylko pierwsze 10 takich poleceń:

```
$ history | grep ls | head
```

możemy też ten wynik zapisać do pliku:

```
$ history | grep ls | head > history.txt
```

a jeśli nie chcemy tracić (nadpisać) dotychczasowej zawartości to skorzystać z `>>` .

Analogicznie do powyższego, do pliku history.txt dopisz 15 ostatnio użytych komend `find`.

Za pomocą polecenia `ps` znajdź na liście procesów linie zawierające nazwy twojej przeglądarki (o ile masz ją uruchomioną).

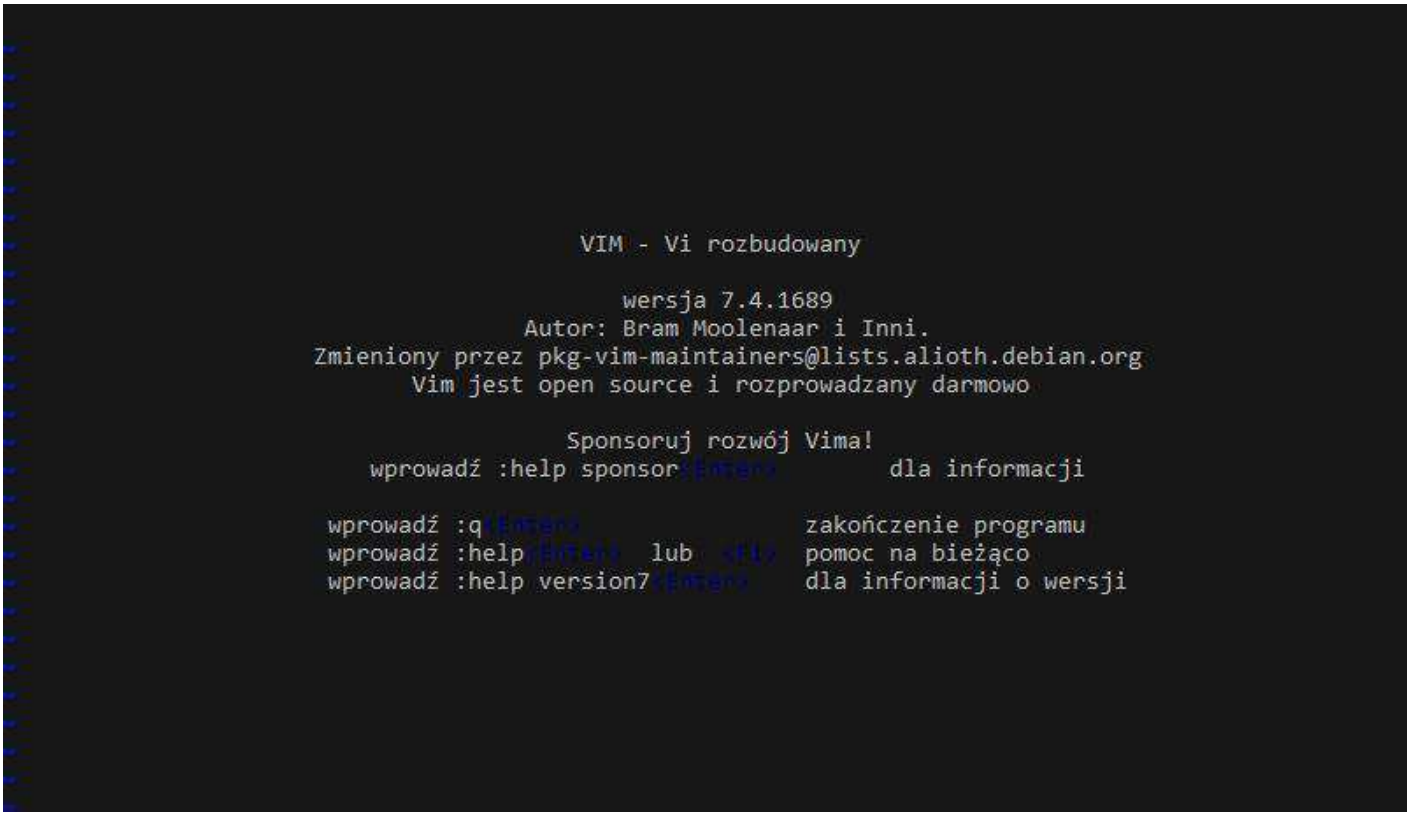
Edytor vi / vim

Co prawda poznaliśmy już podstawowe komendy Linux, warto wspomnieć, że terminal posiada wbudowany edytor – **Vi**. **Vi**oraz jego bardziej popularny klon **Vim**występuje we wszystkich odmianach systemów opartych na UNIXie i od samego początku – czyli lat 70, charakteryzuje się jednolitością implementacji niezależnie od maszyny, na której jest uruchamiany. Oznacza to, że spotkamy go zarówno otwierając terminal Linux, łącząc się z serwerem po ssh, jak i będzie domyślnie dostępny w terminalu Mac i za każdym razem obsługuje się go tak samo. Dzięki temu, **Vim** jest powszechnie używany przez administratorów, zaawansowanych użytkowników i też wielu programistów.

Uruchomienie

Vim uruchamiamy poleceniem `vi` lub `vim`, jednak zdecydowanie wygodniej jest otworzyć tryb edycji podając `vi nazwa_pliku`. Jeżeli plik nie istnieje, to zostanie automatycznie utworzony.

```
$ vi new_file.txt
```



Vim obsługujemy całkowicie z klawiatury. Możemy na razie zapomnieć o myszce. Zaraz po uruchomieniu jesteśmy w trybie poleceń, co dla niektórych może być trochę kłopotliwe. Vim **działa w dwóch trybach – tryb wstawiania (edycji) oraz tryb komend (wpisywania poleceń)**.

Aby z trybu poleceń przejść do trybu edycji – pisania należy nacisnąć na klawiaturze `i`(*inset*) lub `a` (*append*). Polecenia w edytorze Vim nie wymagają zatwierdzenia Enterem, więc po wpisaniu `i` możemy natychmiast zacząć wpisywać dowolne zdanie.

Powrót **z trybu edycji do trybu** komend następuje przez naciśnięcie klawisza **Esc**.

Przykładowe polecenia dostępne w Vimie w trybie poleceń:

- **a** – pisanie za miejscem postawienia kursora
- **i** – wpisywanie tekstu przed kursorem
- **A** – dopisanie na końcu aktualnej linii
- **I** – wdopisanie na początku aktualnej linii
- **o** – dodanie nowej linii poniżej aktualnej (to samo co postawienie kursora na końcu linii i dodanie znaku enter w trybie edycji)
- **O** – dodanie nowej linii powyżej aktualnej (to samo co postawienie kursora na początku lini i dodanie entera)
- **v** – zaznaczenie linii tekstu (v oraz poruszanie się strzałkami ←, →)
- **u** – anulowanie ostatniej zmiany
- **R** – zastępowanie tekstu
- **s** – zastąpienie znaku wskazanego przez kursor
- **S** – zastąpienie aktualnej linii
- **c** – zmiana zaznaczonego tekstu
- **C** – zmiana do końca linii
- **.** – powtórzenie polecenia

- **gg** – powrót do pierwszej linii
- ***/*** – wyszukiewanie miejsca wystąpienia ciągu znaków w tekście

Przetestuj je wszystkie!
(po każdorazowym wykonaniu znowu jesteś w trybie edycji – kliknij Esc, by znowu być w trybie komend)