Linux i Python w Elektronicznej Sieci #04: Więcej na temat Unix'a

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

2023-06-25

1 Operacje na zawartości plików

1.1 grep i wyrażenia regularne

Polecenie `grep [opcje] wyrażenie [plik1 [plik2 [...]]]` wyszukuje pasujące do wyrażenia regularnego wyrażenie linie w plikach, przydatne opcje:
`-v` zamiast pasujących wypisz nie pasujące
`-i` ignoruj wielkość liter
`-a` przetwarzaj pliki binarne jak tekstowe
`-E` korzystaj z „*Extended Regular Expressions*" (ERE) zamiast „*Basic Regular Expressions*" (BRE)

`-P` korzystaj z „*Perl-compatible Regular Expressions*" (PCRE) zamiast „*Basic Regular Expressions*" (BRE)

`-r` rekursywnie przetwarzaj podane katalogi wyszukując w wszystkich znalezionych plikach
`-R` jak -r, ale zawsze podąża za linkami symbolicznymi
`--exclude="wyrażenie"` pomiń pliki pasujące do wyrażenie (może zawierać znaki uogólniające powłoki)

`-l` wypisuje pliki z pasującymi liniami
`-L` wypisuje pliki z bez pasujących linii
`-f` wczytaj wyrażenia z podanego pliku
`-e` może być użyta do poprzedzenia wyrażenia (przydatne zwłaszcza jeżeli chcemy podać kilka)

Wyrażenia regularne[1] konstruuje się w oparciu o następujące znaki specjalne:

```
.      - dowolny znak
[a-z]  - znak z zakresu
[^a-z] - znak z poza zakresu (aby mieć zakres z ^ należy dać go nie na początku)
^      - początek napisu/linii
$      - koniec napisu/linii

*      - dowolna ilość powtórzeń
?      - 0 lub jedno powtórzenie
+      - jedno lub więcej powtórzeń
{n,m}  - od n do m powtórzeń

()     - pod-wyrażenie (może być używane dla powtórzeń, a także referencji
 ↪ wstecznych)
```

Występowanie przełączników `-E` i `-P` wiąże się z ewolucją składni wyrażeń regularnych przy jednoczesnym zachowywaniu kompatybilności z poprzednimi wersjami polecenia grep. Jeżeli coś było traktowane jako zwykły znak nie mogło się tak po prostu stać znakiem specjalnym i należało zastosować zabezpieczanie przy pomocy odwrotnym ukośnikiem lub wybór innego wariantu składni przy pomocy odpowiedniej opcji. W efekcie `grep '^.\?$'` to to samo co `grep -E '^.?$'`, a `grep '^.?$'` to to samo co `grep -E '^.\?$'`.

1. Podana składnia dotyczy „*Extended Regular Expressions*", przy BRE niektóre z znaków sterujących wymagają zabezpieczenia odwrotnym ukośnikiem.

1.2 sed i inne narzędzia przetwarzania tekstów

- `sed [opcje] [pliki]` edytuje plik zgodnie z podanymi poleceniami, przydatne opcje:
  `-e "polecenie"` - wykonuj na pliku polecenie (może wystąpić wielokrotnie celem podania wielu poleceń)
  `-f "plik"` - wczytaj polecenia z pliku plik
  `-E` - używaj rozszerzonych wyrażeń regularnych
  `-i` - modyfikuj podany plik zamiast wypisywać zmieniony na stdout
  `-n` - wyłącza domyślne wypisywanie linii, wypisanie musi być wykonane jawnie poleceniem `p`
  przydatne polecenia:
  `s@regexp@napis@[g]` - wyszukaj dopasowania do wyrażenia regularnego regexp i zastąp je przez napis, podanie opcji g powoduje zastępowanie wszystkich wystąpień a nie tylko pierwszego, znak `@` pełni rolę separatora i może zostać zamiast niego użyty inny znak
  `y@zbiór1@zbiór2@` - zastąp znaki z zbiór1 znakami odpowiadającymi im pod względem kolejności znakami z zbiór2, znak `@` pełni rolę separatora i może zostać zamiast niego użyty inny znak
  możliwe jest też m.in. adresowanie linii na których ma być wykonywana operacja, np: `0,/regexp/s@regexp@napis@` wykona polecenie s na liniach od początku pliku do linii pasującej do wyrażenia regularnego regexp, czyli zastąpi tylko pierwsze wystąpienie w pliku
  Sed jest dość rozbudowanym narzędziem stanowiącym praktycznie coś na kształt interpretera języka programowania (o trochę dziwnej składni) i nie ogranicza się jedynie do prostych przypadków zaprezentowanych w tym skrypcie. Wiele przykładów użytecznych sktyptów sed'owych można znaleźć w spisie dostępnym pod adresem: http://sed.sf.net/sed1line.txt.

- `tail [opcje] [plik]` wyświetla ostatnie linie pliku, przydatne opcje:
  `-n x` określa że ma zostać wyświetlone x ostatnich linii
  `-f` uruchamia dopisywania (gdy do pliku zostaną dopisane nowe linie tail je wyświetli)
- `head [opcje] [plik]` wyświetla początkowe linie pliku, przydatne opcje:
  `-n x` określa że ma zostać wyświetlone x pierwszych linii

- `diff ścieżka1 ścieżka2` porównuje pliki lub katalogi (w przypadku tych drugich porównuje ze sobą pliki o takich samych nazwach oraz zgłasza fakt występowania pliku tylko w jednym z katalogów), przydatne opcje:
  `-r` rekursywnie przetwarzaj podane katalogi
  `-u` wypisuje różnice w formacie "unified"
  `-c` wypisuje różnice w formacie "context"
- `vimdiff ścieżka1 ścieżka2` porównuje pliki wyświetlając je jeden obok drugiego (podobnie jak `diff` z opcją `-y`), pozwalając jednak na edycję tych plików
- `patch` stosuje plik łaty (wynik diff'a) w celu zmodyfikowania plików, typowo:
  `patch -pn < plik.diff` co powoduje zastosowanie zmian opisanych w plik.diff na plikach w bieżącym katalogu, n określa ilość poziomów ścieżek podanych w pliku łaty które mają zostać zignorowane
- `sort [plik]` sortuje linie w wskazanym pliku, przydatne opcje:
  `-n` traktuj liczby jako wartości numeryczne a nie napisy
  `-i` ignoruj wielkość liter
  `-r` odwróć kolejność sortowania
  `-k n` sortuj wg kolumny n
  `-t sep` kolumny rozdzielane są przy pomocy separatora sep

- `cut [opcje] [pliki]` wybiera z pliku zadany zestaw kolumn, przydatne opcje:
  `-f nnn` wypierz kolumny określone przez nnn (np. 1,3-4,6- oznacza kolumnę 1, kolumny od 3 do 4 i od 6, a -3 oznacza 3 pierwsze kolumny)
  `-d sep` kolumny rozdzielane są przy pomocy separatora sep (musi być pojedynczym jedno bajtowym znakiem, aby ominąć to ograniczenie należy skorzystać z awk)
- `paste` łączy (odpowiadające sobie pod względem numerów) linie z dwóch plików

- `join` łączy linie z dwóch plików w oparciu o porównanie wskazanego pola
- `comm` porównuje dwa posortowane pliki pod względem unikalności linii (może wypisać wspólne lub występujące tylko w jednym z plików)
- `uniq` usuwa powtarzające się linie z posortowanego pliku, przydatne opcje:
  `-c` wypisz liczbę powtórzeń
  `-d` wypisz tylko linie z 2 lub więcej wystąpieniami
  `-u` wypisz tylko linie z 1 wystąpieniem

Zadanie 1.0.1

Wyświetl plik `/etc/passwd` z zastąpionym `false` przez `FALSE`.

Zadanie 1.0.2

Polecenie `ls -1d /etc/p*` wyświetla pełne ścieżki do plików i katalogów, znajdujących się (bezpośrednio) w `/etc/`, których nazwa zaczyna się literą p. Przekieruj standardowe wyjście tej komendy do takiego ciągu poleceń aby uzyskać tylko nazwy tych plików (bez ścieżki).

Zadanie 1.0.3

Napisz polecenie które wyszuka wszystkie wystąpienia napisu `nameserver` w plikach znajdujących się w katalogu `/etc` (wraz z jego podkatalogami).

2 Przetwarzanie napisów

2.1 grep, cut, sed, ...

Jako że większość operacji wykonywanych w powłoce takiej jak bash wiąże się z uruchamianiem zewnętrznych programów, to także przetwarzanie napisów może być realizowane w ten sposób. Opiera się na tym jedno z podejść do obsługi napisów w bashu, którym jest korzystanie z standardowych komend POSIX, takich jak grep, cut, sed.

```
# obliczanie długości napisu w znakach, w bajtach i ilości słów w napisie
echo -n "ąbcć 123" | wc -m
echo -n "ąbcć 123" | wc -c
echo -n "ąbcć 123" | wc -w

# obliczanie ilości linii (dokładniej ilości znaków nowej linii)
wc -l < /etc/passwd

# wypisanie 5 pola (rozdzielanego :) z pliku /etc/passwd  z eliminacją
# pustych linii oraz linii złożonych tylko ze spacji i przecinków
cut -f5 -d: /etc/passwd | grep -v '^[ ,]*$'
# komenda cut wybiera wskazane pola, opcja -d określa separator
```

Inną bardzo przydatną komendą jest sed pozwala ona m.in na zastępowanie wyszukiwanego na podstawie wyrażenia regularnego tekstu innym:

```
echo "aa bb cc bb dd bb ee" | sed -e 's@\([bc]\+\) \([bc]\+\)@X-\2-X@g'
```

Sedowe polecenie s przyjmuje 3 argumenty (oddzielane mogą być dowolnym znakiem który wystąpi za s), pierwszy to wyszukiwane wyrażenie, drugi tekst którym ma zostać zastąpione, a trzeci gdy jest g to powoduje zastępowanie wszystkich wystąpień a nie tylko pierwszego.

Należy zwrócić uwagę na różnicę w składni wyrażenia regularnego polegającą na poprzedzaniu `(`, `)` i `+` odwrotnym ukośnikiem aby miały znaczenie specjalne (jeżeli nie chcemy tego robić możemy włączyć obsługę ERE w sed poprzez opcję `-E`).

Innymi przydatnymi komendami przetwarzającymi (specyficznej postaci) napisy są polecaenia `basename` i `dirname`. Służą one do uzyskania nazwy najgłębszego elementu ścieżki oraz ścieżki bez tego najglębszego elementu. Zobacz wynik działania:

```
basename /proc/sys/net/core/
dirname /proc/sys/net/core/
```

Zadanie 2.1.1

Wyświetl z /etc/passwd linie w których UID (3 pole) ma warość >= 1000 nie korzystając z AWK. Jeżeli masz pomysł przedstaw więcej niż jedno rozwiązanie.

Zadanie 2.1.2

Napisz funkcję która przyjmuje dwa argumenty - napis wyszukiwany i napis go zastępujący oraz dokonuje rekurencyjnego wyszukania i zamiany tych napisów w wszystkich plikach w bieżącym katalogu.
Wskazówka 1: polecenie ***`sed`*** *z opcją* ***`-i`*** *i wskazaniem pliku modyfikuje zawartości tego pliku stosownie do poleceń wydanych sed'owi*
Wskazówka 2: dla uproszczenia możesz przyjąć że napisy te składają się jedynie z liter i cyfr.

2.2 awk

Awk jest interpreterem prostego skryptowego języka umożliwiający przetwarzanie tekstowych baz danych postaci `linia=rekord`, gdzie pola oddzielane ustalonym separatorem (można powiedzieć że łączy funkcjonalność komend takich jak grep, cut, sed z prostym językiem programowania).

Wyżej zaprezentowane wypisanie 5 pola (rozdzielanego :) z pliku `/etc/passwd` z eliminacją pustych linii oraz linii złożonych tylko ze spacji i przecinków, realizowane przy użyciu poleceń `cut` i `grep` może być zrealizowane za pomocą samego awk:

```
awk -F: '$5 !~ "^[ ,]*$" {print $5}' /etc/passwd
```

Awk daje duże możliwości przy przetwarzaniu tego typu tekstowych baz danych – możemy np. wypisywać wypisywać pierwsze pole w oparciu o warunki nałożone na inne:

```
awk -F: '$5 !~ "^[ ,]*$" && $3 >= 1000 {print $1}' /etc/passwd
```

Jak widać w powyższych przykładach do poszczególnych pól odwołujemy się poprzez `$n`, gdzie `n` jest numerem pola, `$0` oznacza cały rekord

Program dla każdego rekordu przetwarza kolejne instrukcje postaci `warunek { komendy }`, instrukcji takich może być wiele w programie (przetwarzane są kolejno), komenda `next` kończy przetwarzanie danego rekordu.

Separator pola ustawiamy opcją `-F` (lub zmienną `FS`), domyślnym separatorem pola jest dowolny ciąg spacji i tabulatorów (w odróżnieniu od cut separator może być wieloznakowym napisem lub wyrażeniem regularnym). Domyślnym separatorem rekordu jest znak nowej linii (można go zmienić zmienną RS).

Awk jest prostym językiem programowania obsługującym podstawowe pętle i instrukcje warunkowe oraz funkcje wyszukujące i modyfikujące napisy:

```
echo "aba aab bab baa bba bba" | awk '{
        # dla każdego pola w rekordzie
        for (i=1; i<=NF; ++i) {
                # jeżeli jego numer jest parzysty
                # to zastąp wszystkie ciągi b pojedynczym B
                if (i%2==0)
                        gsub("b+", "B", $i);

                # wyszukaj pozycję pod-napisu B
                ii = index($i, "B")
                # jeżeli znalazł
                # to wypisz pozycję i pod-napis od tej pozycji do końca
                if (ii)
                        printf("# %d %s\n", ii, substr($i, ii))
                # zwróć uwagę że w AWK liczy elementy napisy od 1 a nie od 0
        }
        print $0
}'
```

AWK obsługuje także tablice asocjacyjne pozwala to np. policzyć powtórzenia słów:

```
echo "aa bb aa ee dd aa dd" | awk '
        BEGIN {RS="[ \t\n]+"; FS=""}
        {slowa[$0]++}
        # może być kilka bloków {} pasujących do rekordu
        # jeżeli nie użyjemy next przetworzone zostaną wszystkie
        # {printf("rekord: %d done\n", NR)}
        END {for (s in slowa) printf("%s: %s\n", s, slowa[s])}
'
```

Podobny efekt możemy uzyskać stosując ”uniq -c” (który wypisuje unikalne wiersze wraz z ich ilością) na odpowiednio przygotowanym napisie (spacje zastąpione nową linią, a linie posortowane):

```
echo "aa bb aa ee dd aa dd" | tr ' ' '\n' | sort | uniq -c
```

Jednak rozwiązanie awk można łatwo zmodyfikować aby wypisywało pierwsze wystąpienie linii bez sortowania pliku.

Innym użytecznym zastosowaniem AWK może być wypisanie pliku bez linii pasujących do wzorca oraz linii poprzednich:

```
echo -e "aa\nbb\nWZORZEC\ncc" | awk '{
        # dla linii pasującej do wzorca ustwaiamy flagę print_last na zero i
        ↪ przechodzimy do następnej linii
        /WZORZEC/ {print_last=0; next}
        # jeżeli flaga print_last jest nie zero wypisujemy zapamiętaną poprzednią
        ↪ liniię
        print_last == 1 {print last}
        # zapamiętujemy bierzacą linię do wypisania przy przetwarzaniu kolejnej
        ↪ (jeżeli nie bedzie pasować do wzorca)
        {last=$0; print_last=1}
```

```
        # jeżeli osiągneliśmy koniec pliku i mamy linię do wypisania to ją
          ↪ wypisujemy
        END {if (print_last == 1) print last}
}'
```

AWK pozwala także na definiowanie funkcji:

```
awk 'function f(x) {return 2*x} { print f($1+$2) }'
```

Zadanie 2.2.1

Korzystając z AWK wyświetl z /etc/passwd linie w których UID (3 pole) ma wartość >= 1000.

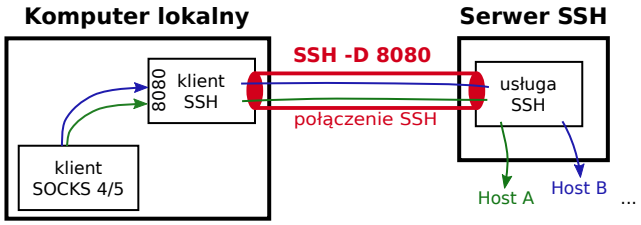
Zadanie 2.2.2

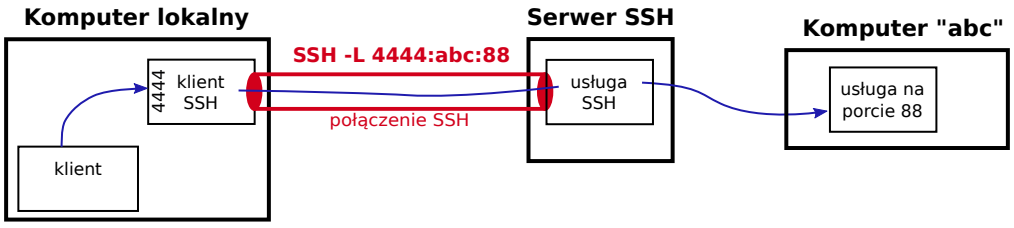
Polecenie `last` wypisuje informację o ostatnich zalogowaniach w systemie. Napisz polecenie (wykorzystujące `last`), które wypisze informację jak często logowali się poszczególni użytkownicy.

3 Praca zdalna

3.1 powłoka zdalna

Komenda `ssh [user@]host` umożliwia uzyskanie powłoki zdalnego systemu poprzez szyfrowane połączenie, przydatne opcje:
`-L portLokalny:hostZdalny:portZdalny` tworzy tunel przekierowujący dane kierowane na portLokalny komputera na którym działa klient ssh do portu portZdalny na serwerze hostZdalny poprzez serwer SSH (przydatne gdy hostZdalny jest osiągalny z hostSSH ale nie z komputera lokalnego)
`-D port` tworzy tunel dynamiczny na wskazanym porcie (może on być użyty jako proxy typu SOCKS np. w Firefoxie w celu zapewnienia dostępu do zasobów WWW dostępnych z serwera SSH a niedostępny z komputera lokalnego)
`-p port` określa inny niż domyślny port serwera SSH
`-X` aktywuje przekazywanie komend X serwera ze strony zdalnej do klienta (pozwala na uruchomienie po stronie zdalnej aplikacji z GUI, które zostanie wyświetlone na lokalnym X serwerze)





3.2 zdalne kopiowanie

Najprostszą metą kopiowania plików pomiędzy różnymi systemami jest wykorzystanie do tego ssh, typowo robi się to na jeden z kilku sposobów:

- poleceniem `scp [opcje] źródło1 [źródło2 [...]] cel` , które kopiuje wskazany plik (lub pliki) do wskazanej lokalizacji, w przypadku kopiowania wielu plików cel powinien być katalogiem, do ważniejszych opcji należy zaliczyć:
  `-r` pozwala na (rekursywne) kopiowanie katalogów
  `-P port` określa port SSH
  W odróżnieniu od `cp` źródło lub cel w postaci `[user@]host:[ścieżka]` wskazują na zdalny system dostępny poprzez SSH.
- poleceniem `rsync [opcje] źródło cel` , które kopiuje (synchronizacjiuje) pliki i drzewa katalogów (zarówno lokalnie jak i zdalnie), do ważniejszych opcji należy zaliczyć:
  `-r` pozwala na (rekursywne) kopiowanie katalogów
  `-l` kopiuje linki symboliczne jako linki symboliczne (zamiast kopiowania zawartości pliku na który wskazują)
  `-t` zachowuje czas modyfikacji plików
  `-u` kopiuje tylko gdy plik źródłowy nowszy niż docelowy
  `-c` kopiuje tylko gdy plik źródłowy i docelowy mają inne sumy kontrolne
  `--delete` usuwa z docelowego drzewa katalogów elementy nie występujące w drzewie źródłowym

  `-e 'ssh'` pozwala na kopiowanie na/z zdalnych systemów za pośrednictwem ssh, źródło lub cel w postaci `[user@]host:[ścieżka]` wskazują na zdalny system `--partial --partial-dir=".-tmp-"` zachowuje skopiowane częściowo pliki w katalogu .-tmp- (pozwala na przerwanie i wznowienie transferu pliku)
  `--progress` pokazuje postęp kopiowania
  `--exclude="wzorzec"` pomija (w kopiowaniu i kasowaniu) pliki pasujące do wzorzec (wzorzec może zawierać znaki uogólniające powłoki) `-n` symuluje pracę (pokazuje co zostałoby skopiowane, ale nie kopiuje)
- stosując `sshfs [opcje] host:scieżka` , który montuje zdalny system plików z użyciem FUSE (filesystem in userspace) oraz SSH, do ważniejszych opcji należy zaliczyć:
  `-p port` określa inny niż domyślny port serwera SSH
  `-o workaround=rename`, który zapewnia poprawne `mv` na istniejący plik
- złożonego polecenia opartego na przekierowaniu wyjścia jakiejś komendy do ssh, które uruchamia po zdalnej stronie proces odbierający te dane na swoim standardowym wejściu, np.:
  - `tar -czf - ścieżka1 [ścieżka2 [...]] | ssh [user@]host 'cat > plik.tgz'` archiwizuje wskazane pliki/katalogi bezpośrednio na zdalny system z użyciem tar i kompresji gzip do pliku `plik.tgz`
  - `tar -cf - ścieżka1 [ścieżka2 [...]] | ssh [user@]host 'tar -xf - -C cel'` kopiuje wskazane pliki/katalogi na zdalny system z użyciem tar do katalogu `cel`

4 Użytkownicy, uprawnienia i procesy

4.1 Uprawnienia do plików

Podstawowe unixowe uprawnienia do plików składają się z trzech członów: uprawnienia dla właściciela (u), grupy (g) i pozostałych użytkowników (o). W każdym z członów mogą być przyznane uprawnienia do czytania (r), pisania (w) i wykonywania (x); w odniesieniu do plików jest to intuicyjne (uprawnienie do wykonywania jest potrzebne do uruchomienia programów), natomiast w stosunku do katalogów wygląda to następująco: uprawnienia do czytania pozwalają na listowanie zawartości, do wykonania pozwalają na dostęp, do zawartości katalogu (wejścia do niego) do pisania na tworzenie nowych obiektów wewnątrz niego i zmienianie nazw istniejących.

Rozszerzeniem podstawowych uprawnień opisanych powyżej jest mechanizm Filesystem Access Control List (ACL, fACL).
Jest on opcjonalnym mechanizmem który (na wspierających go systemach plików) pozwala na definiowanie indywidualnych uprawnień do pliku dla poszczególnych użytkowników i grup – plik ma nadal swojego

właściciela, grupę i wszystkich pozostałych, ale przed prawami dla "others" wchodzą prawa użytkowników i grup definiowanych w ACL. Wypadkowe prawa obliczane są jako suma wynikła z praw użytkownika i grup do których należy.
ACL pozwala ponadto definiować uprawnienia domyślne dla nowo powstałych plików w katalogu (są one opcją katalogu).

Wszystkie poniższe komendy przyjmują opcję `-R` powodującą rekursywne wykonywanie zmian na drzewku katalogów/plików rozpoczynającym się w podanej ścieżce.

- `chown [opcje] właściciel ścieżka` zmiana właściciela pliku
- `chgrp [opcje] grupa ścieżka` zmiana grupy do której należy plik pliku
- `chmod [opcje] uprawnienia ścieżka` zmiana prawa dostępu do pliku(ów)
- `getfacl [opcje] [ścieżka]` odczyt uprawnień związanych z listami kontroli dostępu fACL
- `setfacl [opcje] [ścieżka]` ustawianie uprawnień związanych z listami kontroli dostępu fACL

Dodatkowo należy wspomnieć też o poleceniach takich jak:

- `lsattr` / `chattr` wyświetla / modyfikuje atrybuty plików związanych z systemem plików (np. zabrania jakiejkolwiek modyfikacji pliku)
- `getcap` / `setcap` wyświetla / modyfikuje atrybuty plików związanych z właściwościami jądra (zasadniczo zwiększonymi uprawnieniami programów je posiadających, ale bardziej ograniczonymi niż wykonanie na prawach root przez SUID)

4.2 Użytkownicy

- `id [użytkownik]` informacja o użytkowniku (m.in. grupy do których należy)
- `whoami` informacja o aktualnym użytkowniku
- `w lub who` informacja o zalogowanych użytkownikach
- `passwd [użytkownik]` zmiana hasła
- `su [użytkownik]` przełącza użytkownika (aby przełączony użytkownik miał dostęp do "naszego" x serwera wcześniej wydajemy `xhost LOCAL:użytkownik`)
- `sudo` program pozwalający na wykonywanie uprzywilejowanych komend przez wyznaczonych użytkowników

4.3 Procesy i zasoby

- `ps [opcje]` wyświetla aktualnie działające procesy i informacje o nich
  np. kombinacja opcji `-Af` powoduje wyświetlenie wszystkich procesów w rozszerzonym formacie wypisywania
- `top` monitorowanie procesów obciążających CPU, pamięć, itd
- `iotop` monitorowanie procesów obciążających I/O
- `kill [opcje] pid` przesyła sygnał do procesów o podanych PID
- `killall [opcje] nazwa` przesyła sygnał do procesów o pasującej nazwie

kill i zabijanie procesu

Polecenie kill domyślnie wysyła sygnał SIGTERM, który jest prośbą o zakończenie procesu (proces może ją uszanować lub nie, np. zignorować). Więc sam kill nie zabija procesu.
Wiele sygnałów może zostać przechwyconych i obsłużonych (zignorowanych) przez proces do którego są adresowane. Istnieją także sygnały, które nie mogą zostać obsłużone bądź zignorowane są to m.in.: SIGKILL (zakończenie procesu bez dania mu jakiejkolwiek szansy zrobienia czegoś na „do widzenia", wysyłany przez `kill -9`), SIGSTOP (wstrzymanie procesu).

Ctrl+C / Ctrl+Z / Ctrl+D

Ctrl+C wysyła sygnał SIGINT do procesu zajmującego terminal na którym został on wprowadzony. Sygnał ten jest prośbą o zakończenie procesu, którą proces może uszanować lub nie (np. może całkiem zignorować lub poprosić o potwierdzenie). Jest on podobny do SIGTERM, jednak jest innym sygnałem i może być inaczej obsłużony (np. w SIGTERM nie ma większego sensu pytać o potwierdzenie).

Ctrl+Z wysyła sygnał SIGTSTP do procesu zajmującego terminal na którym został on wprowadzony. Sygnał ten jest prośbą o wstrzymanie procesu i oddanie terminala, prośba ta może być zignorowana przez proces. Proces przerwany w ten sposób może być wznowiony poleceniem `fg` (które wznowi go jako pierwszoplanowy – okupujący terminal) lub `bg` (które wznowi go jako jako proces w tle – oddając terminal, przodkowi który go posiadał wcześniej).

Ctrl+D nie wysyła żadnego sygnału, działa tylko gdy proces czyta dane z terminala (podłączonego zazwyczaj do jego standardowego wejścia). Wysyła on do terminala znak EOT (End-of-Transmission), w efekcie czego:

- (jeżeli bufor wejściowy jest niepusty) terminal wypycha bufor wejściowy do programu (tak jak po wprowadzeniu nowej linii), albo
- (jeżeli nie ma znaków w buforze) terminal zamyka strumień wprowadzanych danych do programu

Program nie otrzymuje w strumieniu znaku EOT (jest on przechwycony przez terminal). Zamknięcie strumienia wejściowego na ogół prowadzi także do zakończenia działania programu, jednak (w odróżnieniu od Ctrl-C) pozwala programowi na normalne przetworzenie wprowadzonych danych.

Ctrl+S / Ctrl+Q

Ctrl+S wstrzymuje przewijanie (odświeżanie) terminala, aby wznowić należy użyć Ctrl+Q.

4.4 Planowanie zadań

Typowo system zapewnia usługę uruchamiania zadań o zadanym czasie. Z usługi tej można skorzystać przy pomocy poleceń:

- `crontab` pozwala oglądać i edytować tablice zaplanowanych zadań cyklicznych (dla cron'a)
- `at` pozwala jednorazowo zaplanować zadanie

Pliki konfiguracyjne crona / obsługiwane crontab-em mają postać: `minuty godzina dzienMiesiaca miesiac dzienTygodnia polecenie`. Wpis oznacza że polecenie ma zostać wykonane jeżeli wszystkie warunki będą spełnione, jeżeli jakiś warunek nie jest nam potrzebny można użyć gwiazdki `*`, z kolei `*/n` oznacza wykonywanie jeżeli dana wartość jest podzielna przez n. Np.: `*/20 3 * * 1 ls` oznacza wykonanie komendy ls w każdy poniedziałek o godzinie 3:00 3:20 i 3:40

Standardowe wyjście, wyjście błędu oraz powiadomienie o niezerowym kodzie powrotu domyślnie są wysyłane na lokalny adres mailowy użytkownika będącego właścicielem danego contaba. Niekiedy dostępny jest także `anacron` pozwalający na mniej precyzyjne planowanie zadań.

5 Wykład wideo[2]

- Operacje na plikach (tekstowych) – http://video.opcode.eu.org/04.01.mkv
- Język AWK – http://video.opcode.eu.org/04.02.mkv
- Praca na systemach zdalnych – http://video.opcode.eu.org/04.03.mkv
- Użytkownicy uprawnienia i procesy – http://video.opcode.eu.org/04.04.mkv

2. Filmy posiadają napisy wgrane do kontenera multimedialnego jako osobny strumień – napisy mogą być włączone lub wyłączone w odtwarzaczu. W wielu filmach dużo dzieje się "na dole ekranu", dlatego polecamy odtwarzać filmy z napisami umieszczonymi poniżej filmu, np. przy pomocy polecenia: `vlc --video-filter='croppadd{paddbottom=120}' --sub-margin=-10 PLIK.mkv`

6 Zadania

Poniższe zadania znajdują się także w odpowiednich rozdziałach skryptu. Zostały jednak zamieszczone zbiorczo także w tym miejscu dla wygody czytelnika.

W rozwiązaniach zadań powinieneś korzystać z poleceń poznanych w ramach dzisiejszych zajęć oraz w ramach poprzednich zajęć z tematyki unixowej. Nie powinieneś stosować w tym celu Pythona, C, C++, itp. chyba że w ramach alternatywnych, nadprogramowych rozwiązań.

Zadanie 1.0.1

Wyświetl plik `/etc/passwd` z zastąpionym `false` przez `FALSE`.

Zadanie 1.0.2

Polecenie `ls -1d /etc/p*` wyświetla pełne ścieżki do plików i katalogów, znajdujących się (bezpośrednio) w `/etc/`, których nazwa zaczyna się literą `p`. Przekieruj standardowe wyjście tej komendy do takiego ciągu poleceń aby uzyskać tylko nazwy tych plików (bez ścieżki).

Zadanie 1.0.3

Napisz polecenie które wyszuka wszystkie wystąpienia napisu `nameserver` w plikach znajdujących się w katalogu `/etc` (wraz z jego podkatalogami).

Zadanie 2.1.1

Wyświetl z /etc/passwd linie w których UID (3 pole) ma warość >= 1000 nie korzystając z AWK. Jeżeli masz pomysł przedstaw więcej niż jedno rozwiązanie.

Zadanie 2.1.2

Napisz funkcję która przyjmuje dwa argumenty - napis wyszukiwany i napis go zastępujący oraz dokonuje rekurencyjnego wyszukania i zamiany tych napisów w wszystkich plikach w bieżącym katalogu.
Wskazówka 1: polecenie *`sed`* z opcją *`-i`* i wskazaniem pliku modyfikuje zawartości tego pliku stosownie do poleceń wydanych sed'owi
Wskazówka 2: dla uproszczenia możesz przyjąć że napisy te składają się jedynie z liter i cyfr.

Zadanie 2.2.1

Korzystając z AWK wyświetl z /etc/passwd linie w których UID (3 pole) ma wartość >= 1000.

Zadanie 2.2.2

Polecenie `last` wypisuje informację o ostatnich zalogowaniach w systemie. Napisz polecenie (wykorzystujące `last`), które wypisze informację jak często logowali się poszczególni użytkownicy.

7 Rozwiązania zadań

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 1.0.1

```
sed -e "s#false#FALSE#g" /etc/passwd
```

Rozwiązanie zadania 1.0.2

```
ls -ld /etc/p* | cut -f3 -d/
```

```
ls -ld /etc/p* | sed -e 's#^/etc/##'
```

Zwróć uwagę na:

- przekierowanie standardowego wyjścia polecenia ls do komendy modyfikującej strumieniowo napisy
- użycie poleceń cut lub sed w roli takiej komendy

Rozwiązanie zadania 1.0.3

```
grep -r nameserver /etc/
```

Zwróć uwagę że:

- do wyszukiwania po zawartości używamy polecenia grep z opcją -r, a nie find
- można by użyć find z warunkiem exec wykonującym grep na danym pliku:
  `find /etc/ -exec grep nameserver \{\} \;`

  ale gdy nie mamy potrzeby stosować innych warunków (dla których potrzebny byłby find) jest to przerostem formy (zarówno w zapisie jak i koszcie obliczeniowym wykonania – find dla każdego pliku musi zrobić fork i exec na odpowiedniego grep'a)

Rozwiązanie zadania 2.1.1

```
while read l; do
    x=$(echo $l| cut -f3 -d:);
    [ $x -ge 1000 ] && echo $l;
done < /etc/passwd
```

```
IFS=":"; while read p1 p2 p3 pX; do
    if [ $p3 -ge 1000 ]; then echo "$p1:$p2:$p3:$pX"; fi
done < /etc/passwd; unset IFS
```

```
# używając cut i paste tworzymy plik złożony z pole 3 tabulator cała linia
cut -f3 -d: /etc/passwd | paste /dev/stdin /etc/passwd | while read uid linia; do
    # następnie czytamy go w pętli while-read jako dwa pola (przed i po
    ↪  tabulatorze)
    [ $uid -ge 1000 ] && echo $linia;
done
```

```
egrep '^([^:]*:){2}[0-9]{4}' /etc/passwd
```

Warto zwrócić uwagę że rozwiązanie z awk (`awk -F: '$3>=1000 {print $0}' /etc/passwd`) jest znacznie prostsze.

Rozwiązanie zadania 2.1.2

```
replace() {
        grep -lR "$1" . | while read f; do
                sed -e "s@$1@$2@g" -i $f;
        done;
}
```

Zamiast `grep -lR "$1" .` można by napisać `grep -R "$1" . | cut -f 1 -d: | uniq`, w większości przypadków zadziała tak samo, ale jest bardziej skomplikowane, wymaga więcej zasobów i będzie miało problem z nazwami plików zawierającymi dwukropek.

Rozwiązanie zadania 2.2.1

```
awk -F: '$3>=1000 {print $0}' /etc/passwd
```

Zwróć uwagę na:

- ustawienie separatora pól przy pomocy opcji -F na dwukropek
- wykorzystanie charakterystyki AWK polegającej na wykonywaniu bloku dla każdej pasującej linii
- warunek na wartość 3 pola, awk sam zadba o konwersję napisu przeczytanego z pliku na liczbę celem wykonania tego porównania

Rozwiązanie zadania 2.2.2

```
last | awk '/^wtmp begins/ {next} $1 i="" {x[$1]++} END{for (u in x) print u,
↪  x[u]}'
```

Zwróć uwagę na:

- zignorowanie linii zaczynającej się od "wtmp begins" z użyciem dopasowania wyrażenia regularnego i komendy next
- zliczanie wystąpień nazwy użytkownika w tablicy x (indeksowanej tymi nazwami)
- zastosowanie bloku END do wypisania zawartości tablicy, której używaliśmy do zliczania użytkowników