SOWID – STRUMIENIE

PAWEŁ JÓZIAK
WYDZIAŁ MATEMATYKI I NAUK INFORMACYJNYCH
POLITECHNIKI WARSZAWSKIEJ

- Nigdy nie wykonuj poleceń metodą *copy&paste*. Przepisz je ręcznie!
- Nie redukuj pracy nad zadaniem do przeczytania go. Wykonaj je i upewnij się, że rozumiesz jak i dlaczego wynik jest taki, jak na ekranie. Zajrzyj do prezentacji bądź zapytaj prowadzącego, jeśli coś nie będzie jasne.
- Linia zaczynająca się od $ reprezentuje prompt zwykłego użytkownika (nie przepisuj go!). Komendą jest to, co następuje po nim.
- Ostrożnie ze spacjami: nie dodawaj ich więcej ani nie usuwaj takowych, jeśli są!

1. Przekierowywanie strumieni

(a) W poniższym przykładzie `cat` czyta z klawiatury (*stdin*), oraz pisze do pliku (*stdout*):
```
$ cat > plik1a
```
Wpisz parę znaków; zakończyć można kombinacją <CTRL> + <D> (pisze znak końca linii) albo <CTRL> + <C> (wysyła sygnał zamknięcia procesu). Sprawdź rezultat:
```
$ cat plik
```

(b) Wyczyść zawartość pliku (tylko zawartość! Plik niech zostanie ten sam) przed nadpisanie go pustym tekstem:
```
$ echo -n > plik1a
$ cat plik1a
```
Być może pamiętasz, że w taki właśnie sposób czyściliśmy historię basha 2 zajęcia temu.

(c) Przykład na różnicę między nadpisaniem a dopisaniem do pliku:
```
$ date > plik1c; date > plik1c; date > plik1c
$ cat plik1c
$ date > plik1c; date >> plik1c; date >> plik1c
$ cat plik1c
```

(d) Trzymanie pliku logów permanentnie otwartego by dodawać kolejne wpisy to zła praktyka. Lepiej dodawać po jednej linii, jak w poniższym przykładzie:
```
$ echo "Pierwszy komunikat" >> plik1d
$ echo "Drugi komunikat" >> plik1d
$ echo "Trzeci komunikat" >> plik1d
$ cat plik1d
```

(e) `'tail'` jest w stanie obsłużyć dopisywanie linii. Utwórz plik i otwórz go przez `'tail -f'`:
```
$ touch plik1e; tail -f plik1e
```
Otwórz drugi terminal i dopisz do niego kilka linii, wywołując kilkukrotnie:
```
$ date >> file1e
```
Obserwuj, co się dzieje w terminalu z `'tail -f'`. Aby go zamknąć: <CTRL> + <C>.

(f) Pliki z poprzednich przykładów już nie będą potrzebne. Aby utrzymać porządek w swoim katalogu domowym:
```
$ rm plik1?
```
(tego jokera już używaliśmy kilkukrotnie, czy wymaga jeszcze wyjaśnień?)

(g) Popraktykuj dodawanie ustawień do pliku konfiguracyjnego. Oto przykład, jak można wyłączyć możliwość odbierania wiadomości czatowych w terminalu:
```
$ echo "mesg n" >> ~/.bashrc
```

(h) Poćwicz ignorowanie wyjścia, ignorowanie błędów oraz ignorowania obu:
```
$ date
$ date > /dev/null
$ date invalid-argument
$ date invalid-argument 2>/dev/null
$ date >/dev/null 2>&1
```
(i) Powrót do znajdowania plików. Podczas poprzednich laboratoriów używaliśmy poniższej komendy:
```
$ find /etc -name "*.conf" -mtime +60 2>/dev/null
```
Jego celem jest ignorowanie komunikatów *"Permission denied"* (i nie tylko), przez przekierowanie *stderr* do 'czarnej dziury'.

2. Natychmiastowe wywołanie

(a) Utwórz plik, którego nazwą będzie dokładna data jego utworzenia:
```
$ touch "$(date)"; ls
```
Zwróć uwagę, że podwójny cudzysłów jest konieczny, inaczej spacje w wyniku wywołania `date` będą interpretowane przez Powłokę. Jeśli to nie do końca jeszcze do Ciebie dociera, spróbuj:
```
$ touch $(date); ls
```
Sprzątanie trzeba wykonać ręcznie.

(b) Wypisanie wszystkich modułów jądra Linuxa:
```
$ find /lib/modules/$(uname -r)/kernel -name "*.ko.zst"
```

(c) A gdybyś zapomniał:
```
$ echo Jestem `id -un`
```

(d) A gdyby była potrzeba zagnieżdżania:
```
$ echo $(echo Jestem $(id -un))
```

3. Filtry

(a) Każdy program wyświetlający treść pliku może być używany jako filtr: `$ set | more`
```
$ set | less
$ set | head
$ set | tail
```

(b) Każdy filtr może być użyty jako samodzielna komenda. Z nieznanych powodów niektórzy używają:
```
$ cat /etc/services | more
```
zamiast po prostu
```
$ more /etc/services
```

(c) Możemy skomponować użycie obu form. Na przykład, siódma linia to ostatnia linia spośród pierwszych siedmiu:
```
$ head -n7 /etc/passwd | tail -n1
```

(d) W Arch Linuksie menedżer pakietów nazywa się `'pacman'` (nie, to nie gra). Zajrzyjmy do zainstalowanych pakietów:
```
$ pacman -Q | less
```

(e) `w`$_{ord}$`c`$_{ount}$ zlicza liczbę linii, słów i znaków:
```
$ wc /etc/passwd
```
Z opcją `'-l'` (albo `'--lines'`) – jedynie linie (a dokładniej: znaki EOL). Zajrzyjmy do przykładów z prezentacji (slajd #17) i policzmy liczbę zmiennych Powłoki:
```
$ set | wc -l
```
liczbę zmiennych środowiskowych:
```
$ env | wc -l
```
plików w bieżącym folderze: `$ ls -A | wc -l`
pakietów zainstalowanych w Arch Linuksie
```
$ pacman -Q | wc -l
```

liczbę użytkowników lokalnych i zdalnych:
`$ getent passwd | wc -l`

(f) `'wc'` jako komenda wypisuje nazwę pliku. W złożeniu strumieni nie ma nazwy pliku, więc czasem może to być wygodniejsze. Porównaj:
```
$ echo "Liczba wpisów w /etc/passwd: $(cat /etc/passwd | wc -l)"
$ echo "Liczba wpisów w /etc/passwd: $(wc -l /etc/passwd)"
```

(g) Domyślnie, zmienne środowiskowe są nieuporządkowane:
`$ env`
ale to nie problem:
`$ env | sort`
Możemy je uporządkować również według wartości:
`$ env | sort -k2 -t=`
(pola rozdzielone `'='`, druga kolumna).

(h) Czytanie z pliku, który jest generatorem losowym, nie jest najmądrzejszym pomysłem:
`$ cat /dev/urandom`
(możesz to przerwać za pomocą <CTRL> <C>). Co więcej, czasem sekwencja niedrukowalnych znaków może wpłynąć na ustawienia wyświetlania naszego terminala. Jeśli tak się stało, możesz zamknąć ten terminal i otworzyć nowy. Wówczas można odpalić tę komendę we właściwy sposób:
`$ cat /dev/urandom | strings`

4. Wyrażenia regularne – podstawy

(a) Jakiego syntaksu dla `'sort'` używaliśmy?
`$ history | grep sort`

(b) Na jakim porcie TCP operuje protokół IMAP4 z SSL (bezpieczny protokół skrzynki pocztowej)?
`$ grep tcp /etc/services | grep imaps`

(c) Czemu przechowywanie sekretów (haseł itp) literalnie w kodzie może nie być najlepszym pomysłem?
`$ strings /bin/bash | grep License`
Porównaj to z:
`$ bash --version`

(d) Adresy IP wydziałowych serwerów mają adresy zaczynające się od '194.29.178'. Niektóre serwisy zostały skonfigurowane, by z nich korzystać. Ale które? Dzięki `'grep'`, skleroza już nie boli:
`$ grep -R "194.29.178" /etc`
Widzimy szereg komunikatów typu *Permission denied*. Oczyśćmy rezultat tej komendy:
`$ grep -R "194.29.178" /etc 2>/dev/null`

(e) `'zgrep'` potrafi operować na plikach skompresowanych:
```
$ cp /etc/passwd .
$ gzip passwd
$ zgrep uszatekm passwd.gz
```

(f) Czy istnieją narzędzia podobne do `'zgrep'`, ale operujące na innych rodzajach archiwów??
`$ ls /usr/bin | grep grep`

(g) Wypiszmy wszystkie linie, które zawierają `'a'` oraz `'c'`, lecz nie zawierają `'u'`:
`$ grep a /etc/passwd | grep c | grep -v u`

5. Wyrażenia regularne POSIX

(a) Zajrzyjmy do przykładowego pliku konfiguracyjnego:
`$ cat /etc/ssh/sshd_config`
(sufiks `'d'` wskazje, że to konfiguracja startu serwera SSH). Tak wygląda typowy plik konfiguracyjny: mnóstwo linii zakomentowanych, część pustych. Czy możemy wyświetlić tylko te wartościowe? Spróbujmy zacząć od tych, które zaczynają się od komentarza:

```
$ grep -vP '^#' /etc/ssh/sshd_config
```
Ale! Jeśli przed # są same znaki białe, to to wciąż jest linia wyłącznie z komentarzam:
```
$ grep -vP '^[ \t]*#' /etc/ssh/sshd_config
```
OK, pozbądźmy się pustych linii – czyli takich, które zawierają najwyżej same białe znaki:
```
$ grep -vP '^[ \t]*$' /etc/ssh/sshd_config
```
A teraz skomponujmy te dwa wyrażenia regularne:
```
$ grep -vP '^[ \t]*(#|$)' /etc/ssh/sshd_config
```

(b) Wyświetl nazwy zmiennych środowiskowych, bez wartości (`'-o'` drukuje jedynie trafienia):
```
$ env | grep -oE '^[^=]*' | sort
```
albo wyłącznie wartości:
```
$ env | grep -oE '[^=]*$' | sort
```

(c) Programy są umiejscowione zazwyczaj w `'/usr/bin'`. Czy istnieje program, którego nazwa zaczyna się i kończy samogłoską?
```
$ ls /usr/bin | grep -P '^[aeiouy].*[aeiouy]$'
```
Ale! Co jeśli program ma nazwę jednoliterową? Potrzebna drobna korekta:
```
$ ls /usr/bin | grep -P '(^[aeiouy].*)?[aeiouy]$'
```
tudzież:
```
$ ls /usr/bin | grep -P '^[aeiouy](.*[aeiouy])?$'
```

(d) Czwarte pole pliku `'/etc/passwd'` zawiera liczbę zwaną GID. Czy istnieją użytkownicy, których GID składa się jedynie z cyfr nieparzystych? Na początek, przypomnijmy sobie jak wygląda rzeczony plik:
```
$ cat /etc/passwd
```
Chcemy zignorować pierwsze 3 pola. Pole najprościej opisać wyrażeniem regularnym `[^:]*:` czyli sekwencja znaków innych dwukropek, wraz z dwukropkiem zaraz po niej. Teraz możemy wywołać:
```
$  grep -P '^([^:]*:){3}[13579]*:' /etc/passwd
```

(e) Czy jacyś użytkownicy mają GID z przedziału 900-999? Pamiętaj: wyrażenie regularne opisuje tekst, nie liczby:
```
$ grep -P '^([^:]*:){3}9..:' /etc/passwd
```
albo lepiej:
```
$ grep -P '^([^:]*:){3}9[0-9]{2}:' /etc/passwd
```

(f) Napisz wyrażenie regularne opisujące prawidłowy adres IPv4. Sprawdź, czy działa:
```
$ grep -RP 'tutaj_twoj_regex' /etc 2>/dev/null
```
i wprowadź poprawki, jeśli to konieczne. To nie jest łatwe zadanie!

(g) Eksperymentuj dalej!