LINUX

o mnie

Adam Bar

Full Stack Developer, Web Enthusiast

https://adambar.pl/
Twitter: @NOtherDev

Co to jest Linux?

I dlaczego się go używa?

> *I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones.*

Linus Torvalds
25.08.1991

Początki Linuksa

- Lata 70. i 80. - UNIX - drogi i niedostępny
- Lata 80. - początek ruchu open-source (GNU)
- 1991 - pierwsza wersja jądra Linuksa
- 1993 - pierwsze dystrybucje
- 2004 - pierwsza wersja Ubuntu

Zastosowania Linuksa

- Serwery
- Systemy wbudowane (embedded):
  - Bankomaty
  - Routery
  - Panele dotykowe
  - Telewizory
  - ...
- Mikrokomputery (Raspberry Pi)
- Urządzenia mobilne: Android, Tizen
- Laptopy i komputery domowe

22%

laptopów/desktopów wśród programistów

67%

serwerów

70%

urządzeń mobilnych (Android!)

Dystrybucje Linuksa

“Linux” to samo jądro. Do użytecznego systemu przyda się jeszcze:

- Powłoka (shell) - np. Bash
- Podstawowe programy
- Menedżer pakietów
- Środowisko graficzne (?)

Wszystko razem to **dystrybucja Linuksa** - np. Ubuntu, Debian, Red Hat, SUSE, Gentoo, Android, Chrome OS… i 500 innych.

(ale macOS to nie Linux - choć jest w znacznej mierze zgodny)

Dlaczego Linux?

Pełna kontrola

Linux niczego nie zabroni.

Zaawansowany system uprawnień użytkowników.

CLI (linia poleceń)

Mnóstwo małych wyspecjalizowanych programów.

Łatwość komponowania programów w funkcjonalne narzędzia.

Automatyzacja

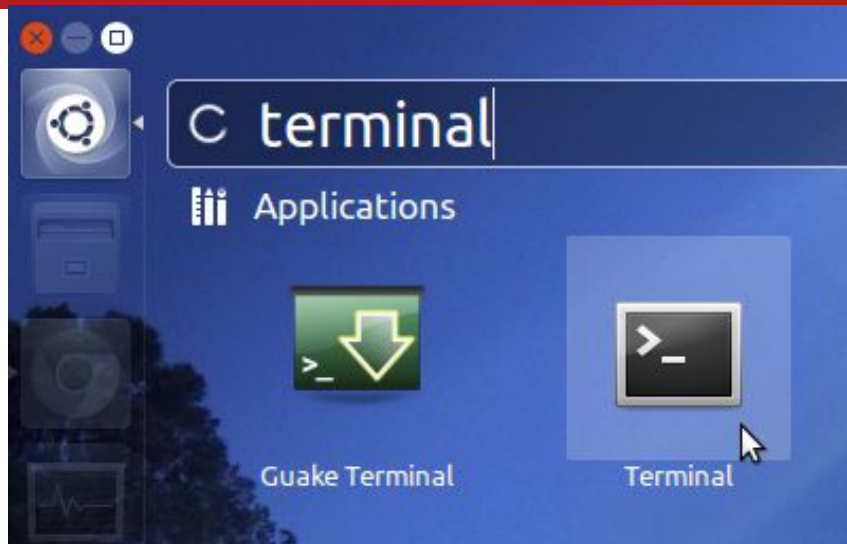
Łatwość tworzenia skryptów dla powtarzających się zadań.

Wbudowane narzędzia uruchamiania wg harmonogramu.

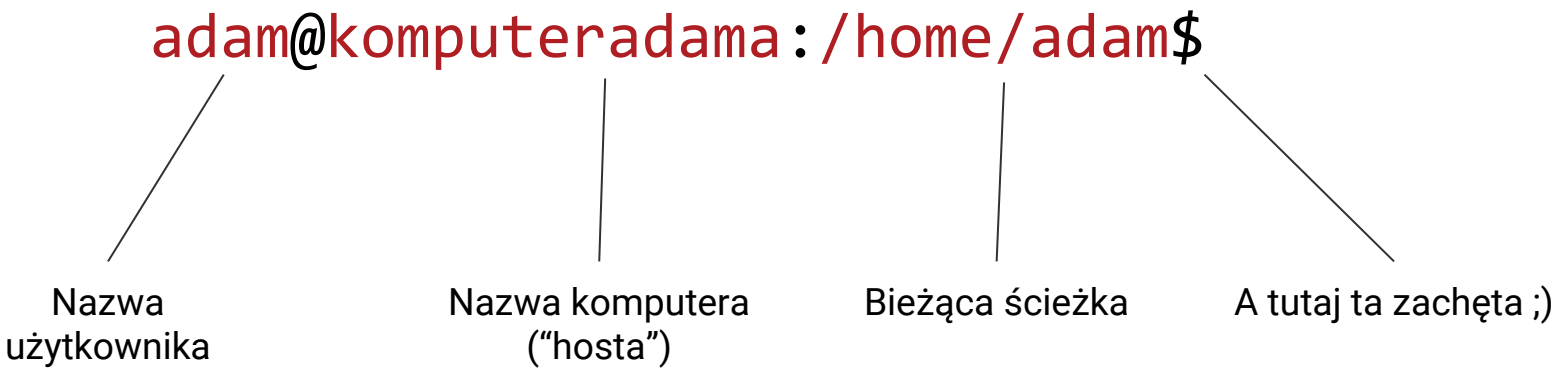
```
docker run --name test-ubuntu -itd ubuntu
docker exec -it test-ubuntu bash
```

terminal

Czyli zapomnij o środowisku graficznym :)



Znak zachęty
linii poleceń



Struktura plików

Struktura katalogów

/ (“root”)
- `/home` - pliki userów
- `/etc` - konfiguracja
- `/dev` - urządzenia
- `/tmp` - pliki tymczasowe
- `/bin` - programy
- `/sbin` - programy systemowe
- `/usr/bin` - programy, znowu
- `/var` - zmienne dane, np. logi

System plików

Rozróżnia wielkość liter:
`Plik.txt` to nie to samo co `plik.txt`!

Nie ma “liter” dysków (C:)

Wszystko jest plikiem

Struktura plików zawiera “specjalne” pliki reprezentujące partycje na dysku, uruchomione procesy, podłączone urządzenia itd.

Dzięki temu wszystko można adresować jednoznaczną ścieżką, np. /dev/sda1

system plików

Struktura katalogów, nawigacja po drzewie

pwd

print working directory

Podaje bieżącą ścieżkę, w której mamy otwarty terminal

```
$ pwd
/home/adam
```

cd <katalog>

change directory

katalogi specjalne:

- `~` domowy
- `-` poprzedni
- `.` bieżący
- `..` nadrzędny

Zmienia ścieżkę w terminalu

```
$ pwd
/home/adam

$ cd /var
$ pwd
/var

$ cd -
$ pwd
/home/adam

$ cd ..
$ pwd
/home
```

ls

list

opcje:
`-l` long
`-a` all

Wypisuje zawartość katalogu (podanego lub bieżącego)

```
$ ls /
bin boot  dev etc home  lib

$ ls -la
total 72
drwxr-xr-x   1 root root 4096 Oct  3 19:42 .
drwxr-xr-x   1 root root 4096 Oct  3 19:42 ..
-rwxr-xr-x   1 root root    0 Oct  3 18:36 .dockerenv
drwxr-xr-x   1 root root 4096 Oct  3 19:37 bin
drwxr-xr-x   2 root root 4096 Apr 10 11:29 boot
drwxr-xr-x   5 root root  360 Oct  7 20:08 dev
```

mkdir <nazwa>

make directory

Tworzy nowy (pod)katalog

```
$ mkdir foo
$ cd foo
$ pwd
/home/adam/foo

$ mkdir /bar
$ cd /bar
$ pwd
/bar
```

`rmdir <nazwa>`

remove directory

Usuwa pusty katalog

```
$ mkdir foo
$ ls
… (jest foo)

$ cd foo
$ ls
$ cd ..
$ rmdir foo
$ ls
… (nie ma foo)
```

cp <z> <do>

copy

opcje:
`-r` recursive - kopiuje podkatalogi

Kopiuje pliki między katalogami

```
$ ls
a.txt

$ cp a.txt /home/hania

$ ls
a.txt
$ ls /home/hania
a.txt

$ cp -r /home/adam /home/jan

$ ls /home/jan
a.txt
```

mv <z> <do>

move

Przenosi pliki między katalogami lub zmienia nazwę

```
$ ls
a.txt

$ mv /home/adam /home/jan

$ cd /home/jan
$ ls
a.txt

$ mv a.txt /home/hania

$ ls
$ ls /home/hania
a.txt
```

`rm <plik>`

remove

opcje:
`-r` recursive - usuń także z podkatalogów
`-f` force - wiem co robię!

Usuwa pliki

```
$ ls
a.txt

$ rm a.txt
$ ls
(pusto!)

$ rm -rf /tmp/bzdury
$ cd /tmp/bzdury
No such file or directory

$ rm -rf /
(koniec pracy w tym systemie :))
```

Ćwiczenie 1 - konkurs :)

Znajdź pliki znajdujące się możliwie najgłębiej w hierarchii katalogów w Twoim systemie.

Czy wiesz, że?
Autouzupełnianie: naciśnij `tab` po wpisaniu fragmentu nazwy katalogu

apt-get

Menedżer pakietów w Ubuntu

apt-get update
apt-get install <pakiet>

Advanced Packaging Tool

Zarządza pakietami (oprogramowaniem) dostępnym w systemie

```
$ less a.txt
command not found
```

```
$ apt-get update
```
(odświeża listę źródeł)

```
$ apt-get install less
```
(instaluje pakiet less)

```
$ less a.txt
```
(ok!)

curl <adres>

wget <adres>

Dwa konkurencyjne narzędzia do pobierania z sieci - oba z ogromną ilością opcji.

`curl` domyślnie wyświetla zawartość na ekran, nie podąża za przekierowaniami.
`wget` domyślnie zapisuje zawartość do pliku, podąża za przekierowaniami.

```
$ curl wp.pl
<html>
<head><title>301 Moved
Permanently</title></head>
(...)

$ wget wp.pl
(...)
'index.html' saved [299799]
```

pliki cd.

Wyświetlanie zawartości plików

`cat <plik>`

conCATenate...

Wypisuje plik/pliki

```
$ cat adam.txt
Adam Bar
```

head <plik>

Wypisuje pierwsze 10 linii pliku

```
$ head process.log
(...)
```

tail <plik>

opcje:
`-f` follow - śledzi zmiany na bieżąco

Wypisuje ostatnie 10 linii pliku

```
$ tail server.log
(...)

$ tail -f server.log
(...)
```

`less <plik>`

“less is more” :)

Wypisuje pliki fragment po fragmencie.

W odróżnieniu od prostszych narzędzi, pozwala na poruszanie się po pliku za pomocą strzałek, Page Up/Down oraz wyszukiwanie

```
$ less process.log
(...)
```

`/igła` - przechodzi do pierwszego wystąpienia “igła”
`n` - przechodzi do kolejnego wystąpienia
`g` - przechodzi do początku pliku
`G` - przechodzi na koniec pliku
`F` - tryb “follow”
`q` - wyjście

Ćwiczenie 2

Przeglądanie pliku

https://raw.githubusercontent.com
/NOtherDev
/zadania-linux
/master/cwiczenie-2.txt

1. Pobierz plik z GitHuba za pomocą terminala - bez użycia gita
2. Otwórz plik w terminalu - dowolną poznaną metodą (być może warto doinstalować `less`-a?)
3. Przejdź na koniec pliku i zapisz ostatnie słowo - to początek hasła
4. Przejdź na początek pliku
5. Wyszukaj wszystkie wystąpienia słowa “hasło” i każdorazowo zapisz słowo znajdujące się bezpośrednio po nim
6. Daj znać, gdy skompletujesz całe hasło

edytory tekstu

Modyfikacja zawartości plików

`vim <plik>`

czyli “jak stąd wyjść”?

Zaawansowany edytor tekstowy
z ”intuicyjnymi” poleceniami.

Posiada tryb poleceń (domyślnie uruchomiony) i tryb wstawiania (pisania). Do trybu wstawiania wchodzi się klawiszem `i` (insert). Do trybu poleceń wraca się klawiszem `Esc`.

Przydatne polecenia:

`:q`	wyjście
`:q!`	wyjście bez zapisu, mimo zmian
`:wq`	wyjście z zapisem
`dd`	usuń linię
`/term`	szukaj wystąpienia słowa “`term`”

`nano <plik>`

`mcedit <plik>`

Bardziej “przyziemne” edytory tekstu.

`nano` jest lekki i dość często spotykany. Zawiera podstawowy zestaw komend, opisanych w pasku na dole ekranu, dostępnych z klawiszem `Ctrl`.

`mcedit` to część Midnight Commandera - dużego “okienkowego” menedżera plików. Raczej niespotykany na serwerach...

Ćwiczenie 3

Modyfikowanie plików

https://raw.githubusercontent.com
/NOtherDev
/zadania-linux/master
/cwiczenie-3.txt

1. Pobierz plik z GitHuba za pomocą terminala - bez użycia gita
2. Skopiuj plik dwukrotnie i pozmieniaj nazwy, w taki sposób, aby mieć pliki:
   - vim.txt
   - nano.txt
   - mcedit.txt
3. Otwórz kolejno pliki za pomocą odpowiednich edytorów, wg ich nazw
4. Wykonaj polecenia znajdujące się w plikach
5. Wybierz swój ulubiony edytor ;)

uprawnienia

Użytkownicy, grupy, prawa dostępu

Użytkownicy
a Grupy

Użytkownik

- loguje się do systemu
- ma katalog domowy `/home/<user>`
- właściciel plików

```
whoami
adduser <user>
su <user> (switch user)
sudo <polecenie>
```

Grupa

- może zawierać wielu użytkowników
- użytkownik może należeć do wielu grup

```
groups [user]
addgroup <group>
adduser <user> <group>
```

Trzy poziomy uprawnień

Odczyt (r = read) **4**

Zapis (w = write) **2**

Uruchomienie (x = execute) **1**

tylko odczyt	4 + 0 + 0	**4**	`r--`
odczyt + zapis	4 + 2 + 0	**6**	`rw-`
odczyt + uruchomienie	4 + 0 + 1	**5**	`r-x`
pełen dostęp	4 + 2 + 1	**7**	`rwx`

Trzy podmioty uprawnień

Dla właściciela (u = user) **Dla grupy (g = group)** **Dla pozostałych (o = others)**

pełen dostęp, ale tylko dla właściciela	**700**	`rwx------`
pełen dostęp dla właściciela i grupy	**770**	`rwxrwx---`
pełen dostęp dla właściciela, reszta tylko odczyt	**744**	`rwxr--r--`
zapis dla właściciela, odczyt i uruchomienie dla wszystkich	**755**	`rwxr-xr-x`
odczyt + zapis dla wszystkich	**666**	`rw-rw-rw-`

chmod <mode> <plik>

change mode

opcje:

`-R` recursive - zmienia wszystkim plikom w (pod)katalogach

Ustawia lub modyfikuje uprawnienia do pliku

```
$ chmod 0777 a.txt
$ ls -l
```
(w uproszczeniu)
```
rwxrwxrwx a.txt

$ chmod -x a.txt
$ ls -l
rw-rw-rw- a.txt

$ chmod u+x a.txt
$ ls -l
rwxrw-rw- a.txt
```

chown <user> <plik>

change owner

opcje:

`-R` recursive - zmienia wszystkim plikom w (pod)katalogach

Modyfikuje właściciela pliku

```
$ la -l
(w uproszczeniu)
root root a.txt

$ chown adam a.txt
$ ls -l
adam root a.txt

$ chmod -R adam /tmp/pliki
$ ls -l /tmp/pliki
adam root plik1
adam root plik2
```

chgrp <grupa> <plik>

change group

opcje:

`-R` recursive - zmienia wszystkim plikom w (pod)katalogach

Modyfikuje grupę pliku

```
$ la -l
(w uproszczeniu)
root root a.txt

$ chgrp members a.txt
$ ls -l
root members a.txt

$ chgrp -R members /tmp/pliki
$ ls -l /tmp/pliki
root members plik1
root members plik2
```

Ćwiczenie 4

utworzenie pustego pliku:
`touch <plik>`

1. Stwórz trzy konta, nazwane:
   a. swoim imieniem
   b. innej osoby z grupy
   c. osoby z innej grupy
2. Stwórz odpowiednie grupy w systemie
3. Przypisz konta osób do odpowiednich grup
4. Z konta nazwanego swoim imieniem stwórz plik `sekret.txt` i nadaj pełne uprawnienia tylko dla siebie
5. Stwórz plik `dlagrupy.txt` i pozwól swojej grupie na odczyt - zapis tylko dla siebie
6. Stwórz plik `tablica.txt` i pozwól wszystkim na pełen dostęp
7. Sprawdź, co mogą zrobić inne osoby!

sort

uniq

wc

word counter

opcje:
`-l` lines

```
$ sort
2
1
(Ctrl+D)
1
2

$ uniq
adam
adam
(Ctrl+D)
adam

$ wc -l
Litwo, Ojczyzno moja
Ty jestes jak zdrowie
(Ctrl+D)
2
```

strumienie, potoki

Hydrologia?

Strumienie
stdin/stdio streams

Strumień wejściowy (stdin) - dane od użytkownika

- domyślnie: dane z klawiatury
- często: strumień z pliku

```
$ sort
granat
awokado
(Ctrl+D)
awokado
granat
```

```
$ sort < owoce.txt
awokado
granat
```

Strumienie
stdin/stdio streams

Strumień wyjściowy (stdout) - wynik programu

- domyślnie: wypisywanie na terminal
- często: strumień do pliku (stworzenie/nadpisanie zawartości)

```
$ echo “plynie Wisla plynie”
plynie Wisla plynie

$ echo “plynie Wisla plynie” > zbiornik.txt
$ cat zbiornik.txt
plynie Wisla plynie
```

Strumienie
stdin/stdio streams

Strumień wyjściowy może także dopisywać zawartość do pliku:

```
$ echo “po polskiej krainie” >> zbiornik.txt
$ cat zbiornik.txt
plynie Wisla plynie
po polskiej krainie
```

Potoki (pipe) -
komunikacja między programami

Strumienie można przekazywać między programami bez pośrednictwa plików:

```
$ cat liczby.txt | sort | uniq | wc -l
12
```

(wynik ostatniej komendy - liczba linii, po wcześniejszym posortowaniu i usunięciu duplikatów z pliku `liczby.txt`)

Logiczne łączenie komend ("wykonaj, jeśli")

Polecenia można łączyć operatorami logicznymi

```
$ mkdir test && echo "moge pisac" > test/test.txt
```
(jeżeli użytkownik nie ma uprawnień do stworzenia katalogu, `mkdir` zwróci błąd i polecenie `echo` nie uruchomi się)

```
$ rm tmp; echo "nie zwazaj na wynik"
$ rm tmp || echo "tu takze"
```
(echo będzie wywołane niezależnie, czy rm zwróci błąd, czy nie)

pomoc

opcji jest zawsze więcej, niż uda się zapamiętać

`<program> -h`
`<program> --help`

`man <program>`
manual

Programy zazwyczaj udostępniają skróconą informację o użyciu po uruchomieniu z parametrem `-h` lub `--help`

```
$ less --help
```

Dodatkowo, standardowo, pełniejsze instrukcje wyświetla program `man`

```
$ man less
```

inne narzędzia

przydatne w pracy na Linuksie

Czy wiesz, że?
Możesz przerwać działanie większości programów naciskając kombinację `Ctrl+C`.

find

```
find
```

```
find <sciezka>
```

```
find <sciezka>
 -name <nazwa>
```

Wypisuje wszystkie pliki spełniające zadane kryteria

```
$ find .
./zbiornik.txt
./owoce.txt
./stare/adam.txt
```

```
$ find / -name README
/etc/terminfo/README
/etc/sysctl.d/README
/etc/alternatives/README
/etc/init.d/README
(...)
```

grep <igla>

grep -e <regex>

grep -v <wykluczenie>

Filtruje linie z wejścia i zwraca jedynie pasujące do poszukiwanego wyrażenia

```
$ cat liczby.txt | grep 1
1
10
11
12
(...)

$ cat liczby.txt | grep -e ‘[123]2’
12
22
32

$ cat liczby.txt | grep -v 1
2
3
(...)
```

Ćwiczenie 5

Składanie poleceń

Za pomocą kompozycji poleceń
i przekierowań strumieni:

1. Znajdź w systemie wszystkie pliki, których nazwa zawiera słowo “linux” i zapisz je do pliku “pliki.txt”
2. Znajdź w systemie wszystkie pliki, których nazwa zawiera słowo “gnu” i **dopisz** je do pliku “pliki.txt”
3. Policz ilość unikatowych linii w pliku “pliki.txt”

df -h

Oblicza rozmiar zajętej i wolnej przestrzeni w systemie plików. Opcja `-h` oznacza “human-readable” (dotyczy jednostek rozmiaru)

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          63G   38G   22G  64% /
```

du -sh
du <katalog>

Oblicza rozmiar plików w danym katalogu. Opcja `-s` oznacza “summarize” (zlicza wszystkie podkatalogi, zamiast wypisywać je osobno)

```
$ du -sh ~
260K    .
```

ps aux

top

Dwie metody na podejrzenie listy procesów działających aktualnie w systemie (posiadają wiele opcji).

```
$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  18236  2880 pts/0    Ss+  Oct09   0:00 /bin/bash
root        11  0.0  0.1  18312  3368 pts/1    Ss   Oct09   0:00 bash
root      7073  0.0  0.1  34424  2912 pts/1    R+   15:19   0:00 ps aux
```

ssh <user>@<adres>

Połączenie ze zdalną maszyną udostępniającą linię poleceń (shella) - np. z serwerem.

```
$ ssh adam@192.168.1.1
$ ssh deployer@prod-in-cloud
```

scp <zrodlo>: <plik> <plik>

scp <plik> <cel>:<plik>

Narzędzie do kopiowanie plików ze zdalnych maszyn lub na zdalne maszyny - “pod spodem” używa połączenia `ssh`.

```
$ scp web@server:/var/logs.tar ~/logi
```
(z serwera na lokalną maszynę)

```
$ scp config web@server:/etc/config
```
(z lokalnej maszyny na serwer)

ln -s <cel> <nazwa>

Tworzy symboliczne dowiązanie (alias) do pliku. Bogatszy odpowiednik skrótów z Windowsa.

Z punktu widzenia większości programów, dowiązanie jest identyczne z docelowym plikiem:

- można je edytować (zmianie ulegnie docelowy plik),
- można je przenosić/kopiować (przeniesiona zostanie faktyczna zawartość, nie “skrót”)

```
$ ln -s /bin/inny /usr/bin/program

$ ls -l /usr/bin/program
lrwxrwxrwx [...] program -> /bin/inny
```

cut -d <separator> -f <kolumny>

Narzędzie do wycinania fragmentów z linii tekstu. Typowe użycie - wycinanie kolumn z pliku comma-separated.
Parametr `-d` specyfikuje separator rozdzielający kolumny.
Parametr `-f` specyfikuje numer(y) kolumn, które mają być wybrane.

```
$ cat dane.csv
Jan,Kowalski,1976-05-25,M
Anna,Nowak,1990-04-21,F

$ cat dane.csv | cut -d ’,’ -f 2,4
Kowalski,M
Nowak,F
```

`tar -cf <archiwum> <pliki...>`
(archiwizacja)

`tar -xf <archiwum>`
(rozpakowanie)

tape archive

Łączy wiele plików w jeden plik “archiwum” - gotowy do kompresji

```
$ tar -cf archiwum.tar plik1 plik2
```
(utworzy plik archiwum.tar)

```
$ ls
archiwum.tar plik1 plik2

$ rm plik*

$ tar -xf archiwum.tar
```
(odtworzy pliki plik1 i plik2)

```
$ ls
archiwum.tar plik1 plik2
```

gzip <archiwum>
(kompresja)

gunzip <archiwum>
(rozpakowanie)

Kompresuje plik archiwum

```
$ gzip archiwum.tar
$ ls
archiwum.tar.gz

$ gunzip archiwum.tar.gz
$ ls
archiwum.tar
```

Można też użyć bezpośrednio opcji `tar -z`

```
$ tar -czf arch.tar.gz plik1 plik2
$ rm plik*
$ ls
arch.tar.gz

$ tar -xzf arch.tar.gz
$ ls
arch.tar.gz plik1 plik2
```

Ćwiczenie 6

Parsowanie logów

https://raw.githubusercontent.com/NOtherDev/zadania-linux/master/cwiczenie-6.tar.gz

1. Pobierz plik archiwum z GitHuba - z terminala, bez użycia gita
2. Rozpakuj pliki z archiwum
3. Zapisz do pliku “strony.txt” wszystkie adresy, które odwiedzili użytkownicy telefonów Samsung SM-G950F.
   Wpisy ze wszystkich trzech dni umieść łącznie w jednym pliku, posortowane i wyłącznie unikatowe

cron

harmonogram zadań

Wyrażenia harmonogramu

cron

Wyrażenia określają, kiedy wykonana zostanie dana komenda.

minuta	godzina	dzień	miesiąc	dz. tyg.	
Przykłady					
15	*	*	*	*	raz na godzinę - 0:15, 1:15...
*	*/2	*	*	1-5	co minutę w godziny parzyste, ale tylko pon-pt
0	12	1,11,21	*	*	1, 11 i 21 dnia miesiąca w południe

`crontab -e`
edycja harmonogramu

`crontab -l`
podgląd harmonogramu

Polecenia w harmonogramie ustawia się poprzez edytor tekstowy uruchamiany przez `crontab -e`.

Składnia:

```
# komentarz
<wyrazenie>    <polecenie>
```

np.

```
# eksport logow codziennie o 4:00
0 4 * * *  /export-logs.sh
# sesje bankowe 3x w dni powszednie
0 7,12,15 * * 1-5  /process.sh >> logs.log
# healthcheck co minute
* * * * *  /healthcheck.sh
```

Ćwiczenie 7

Harmonogram zadań

Wypisanie dwóch wyników naraz:
`$ echo “`date`: `users`”`
` - “backtick” - tam, gdzie “~”

1. Ustaw w systemie zadanie, które co dwie minuty dopisze do pliku “czuwak.txt” informację jak długo system jest uruchomiony (polecenie `uptime`) oraz liczbę procesów w systemie.
2. Ustaw w systemie zadanie, które co dwie godziny pobierze zawartość strony głównej któregoś z portali informacyjnych i jeśli odpowiedź zawiera słowo “Lewandowski”, dopisze do pliku “wzmianki.txt” bieżącą datę (polecenie `date`).

git

surowy i “prawdziwy”

```
git clone <repo>
git pull
git status
git branch <branch>
git checkout <branch>
git add <pliki>
git commit -m <opis>
git push
git merge <branch>
git rebase -i <branch>
git log
git ...
```

Każda operacja w gicie posiada osobną stronę z pomocą i opcjami:

```
$ git help rebase
```

Aliasy git

Git umożliwia tworzenie własnych nazw do złożonych komend - aliasów - poprzez edycję pliku `~/.gitconfig` lub komendę:

```
git config --global alias.<alias> <komenda>
```

Niezwykle użyteczne aliasy:

```
alias.lola=log --color --graph --pretty=format:'%Cred%h%Creset
-%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset'
--abbrev-commit --al

alias.please=push --force-with-lease
```

Ćwiczenie 8

git w konsoli

1. Sklonuj dowolne repozytorium z GitHuba za pomocą linii poleceń
2. Załóż nowy, własny branch
3. Wprowadź dowolną zmianę (również za pomocą linii poleceń)
4. Wyślij zmianę do zdalnego repozytorium (do swojego forka)

Terminal to
Twój przyjaciel

...szczególnie kiedy dodatkowo ułatwia pracę.

https://github.com/robbyrussell/oh-my-zsh

- pluginy podpowiadające składnie mnóstwa poleceń
- wizualne wskaźniki statusu gita i aktualnego brancha
- wskaźnik wyniku ostatniej komendy
- wygodne przeszukiwanie historii poleceń

dzięki!

Adam Bar

https://adambar.pl/
Twitter: @NOtherDev

Skrypty
w Bashu

Najprostsze skrypty w Bashu to zestawy komend zebrane w pliku *.sh

```
$ cat date.sh
echo “Dzisiaj jest: `date`“
$ chmod +x date.sh
$ ./date.sh
Fri Oct 13 15:00:00 UTC 2017
```

Zmienne
w Bashu

```
DATA=`date`
echo “Dzisiaj jest $DATA“

LICZNIK=1
LICZNIK=$(($LICZNIK+1))
echo $LICZNIK    # wynik: 2

echo $1          # pierwszy parametr wywolania skryptu
```

Zmienne systemowe (można je podejrzeć przez `env`):
`PATH, HOME, PWD...`

Warunki
w Bashu

```
if [ $1 -gt 100 ]
then
  echo “Podano duza liczbe!”
fi

if [ -e $2 ]
then
  echo “Plik $2 istnieje!”
else
  echo “Plik $2 nie istnieje :(“
fi
```

Pętle
w Bashu

```
while [ $counter -le 100 ]
do
  echo $counter
  (($counter++))
done

for file in *.html
do
  diff file _base.html
done
```