Narzędzia Wspierające Programowanie

Bash – poszerzenie podstaw

- Praca z katalogami. Włączmy terminal.
- Wypis zawartości katalogu : można go ukierunkować.

Samo `ls` wypisuje tylko nazwy, a `ls -l` ma długi wypis i słabo rozróżnia rodzaje obiektów.

Wypis nazw, 1 nazwa na linię: `$ ls -1` (przyda się w seryjnej obróbce danych)

Wypiszmy tylko ścieżki: `$ ls -d */`

→ Propozycja: `$ ls -ogh --color`

Krótszy wypis, czytelne długości plików, koloryzacja ścieżek, ...
(*komenda jest długa – zaradzimy temu przy konfiguracji bash'a)*

- Gimnastyka w katalogach

Utwórzmy najpierw strukturę ścieżek i plików w nich:

```
$ cd ~ ; pwd                          ←  ~ to katalog domowy
$ mkdir code_draft code code_backup
$ cd code_draft
$ touch code0_draft.C                 ←  touch tworzy pusty plik
$ cd ../code
$ mkdir src include bin
$ touch src/code1.C src/code2.C
$ touch include/header.h
$ tree ~/code*                        ←  tree wyświetla foldery/ścieżki jak drzewo
```

Teraz poćwiczmy kopiowanie, przenoszenie i kasowanie:

```
$ pwd; cd src
$ cp -p code1.C ../../code_backup     ←  cp -p kopiuje, zachowując atrybuty (np. datę)
$ tree ~/code*
$ cp -pr ../include ../../code_backup/          ←  cp -r  kopiuje ścieżki
$ tree ~/code*
$ mv -f ../../code_draft/code0_draft.C ./code0.C    ←  mv -f  przenosi ścieżki
$ tree ~/code*
$ rm -r ~/code_draft                  ←  rm -r  kasuje ścieżkę (zawartość też)
```

Narzędzia Wspierające Programowanie

Bash – poszerzenie podstaw

⊕ *Dodatek*. Sprawdźmy podstawowe cechy i zasoby.

Ilość pamięci:	`$ free -h`	
Miejsce na dysku:	`$ df -H .`	
Miejsce na koncie:	`$ quota -s -u {MyLogin}`	
Nazwa dystrybucji Linuxa:	`$ cat /etc/issue`	(lub: `hostnamectl`)
Ilość wątków CPU:	`$ nproc`	
Adres DNS komputera:	`$ hostname`	
Adres IP komputera:	`$ hostname -I`	
Informacje o CPU:	`$ less /proc/cpuinfo`	(wyjście: `q`)

⊕ *Dodatek*

Wypis plików ukrytych:	`$ ls -a`	(pliki o nazwach zaczynających się od `.`)
Kopiowanie z paskiem postępu	`$ gcp`	(przy dużych danych – szacujemy postęp)

⦿ Poszukiwanie plików

Aby wyszukać plik(i): `$ find {od/sciezki} -name {plik/i}`

Np.:

```
$ cd Narzedzia_Wspierajace_Programowanie/MathTools
$ find . -name "*.C"
```

Aby wyszukać ścieżkę: `$ find {od/sciezki} -type d -name {ścieżka/i}`

⦿ Wyszukiwanie fraz w środku plików

Aby wyszukać frazy: `$ grep {fraza} {plik}`

Często jednak szukamy frazy, nie wiedząc, w którym jest pliku, a np. jest wiele podkatalogów.

💡 Propozycja: `$ grep -inr --color --include=*.{C,h} szukana_fraza`

gdzie: `-r (recursive)`	szukaj w podkatalogach
`-i (ignore case)`	akceptuj każdą wielkość liter
`-n (line number)`	wypisz nr linii, gdzie jest fraza
`--include={type}`	szukaj wśród plików tego typu

(*komenda jest długa – zaradzimy temu przy konfiguracji bash'a)*

Np.:

```
$ grep Status *
$ grep --color -inr --include=*.{C,h} Status
```

⦿ Linki symboliczne

Jeśli plik ma występować w kilku katalogach lub pod kilkoma nazwami,
to kilka kopii tego samego - marnotrawi zasoby na dysku.
Lepiej utworzyć link symboliczny do oryginału.

Link do pliku, adres względny: `$ ln -s {ścieżka_stąd}/plik_oryginalny .`

Podanie pełnej lokalizacji: `$ ln -s /pelna/ścieżka/do/plik_oryginalny .`

Wypis linku i jego źródła: `$ ls -l {link_symboliczny}`

Np.

```
$ ln -s Narzedzia_Wspierajace_programowanie/Smash/run1/outdir/particle_lists.oscar .
$ ls -l particle_lists.oscar
lrwxrwxrwx 1 kpias kpias 74 lut 26 09:12 particle_lists.oscar →
Narzedzia_Wspierajace_Programowanie/Smash/run1/outdir/particle_lists.oscar
```

⊕ *Dodatki*

- Wypis tylko ścieżek w katalogu: `$ ls -l | grep ^d`
- Wypis tylko linków w katalogu: `$ ls -l | grep ^l`

- Jak szybciej pisać w terminalu dzięki skrótom klawiszowym:

[Ctrl Shift C/V]	copy/paste w terminalu
[Ctrl ←/→]	w lewo/prawo o słowo
[Ctrl A]	skok na początek linii
[Ctrl E]	skok na koniec linii
[Ctrl U]	kasuj od kursora w lewo
[Ctrl K]	kasuj od kursora w prawo
[Ctrl D]	kasuj znak pod kursorem
[Ctrl W]	kasuj od kursora do początku słowa
[Alt D]	kasuj od kursora do końca słowa
[Ctrl T]	zamień literę pod kursorem z poprzednią
[Ctrl R]	podpowiedzi z poprzednich komend

Szersze zestawienie skrótów klawiszowych i pomocniczych komend np. tu: Link

⦿ less: czytnik plików tekstowych w terminalu

Często chcemy na szybko przejrzeć plik (np. z danymi – wtedy plik może być b. długi).
Wciąganie długiego pliku do edytora długo trwa ⇒ Warto znać szybkie czytniki tekstowe.
Również, czytnik zabezpiecza przed nieumyślnym nadpisaniem.

`$ less {plik}` : Szybki (nawet b. długi plik otworzy się w moment)
Brak koloryzacji składni kodów

`less -N {plik}` włącz z numeracją linii

`PgDn` , `PgUp`	skok do następnej / poprzedniej strony
`G`, `g`	skok na koniec / początek pliku
`/fraza`	szukaj wystąpienia frazy
	`n` , `N` szukaj następne/poprzednie wystąpienie
`q`	wyjście

⦿ nano: edytor plików tekstowych w terminalu

Edytor prosty w obsłudze. Koloryzacja składni kodów. Możliwość edycji kilku plików na raz.

💡 Propozycja: `$ nano -FP {plik}`

gdzie: `-P` zapaamięta pozycję w pliku z ostatniej edycji
`-F` umożliwi edycję wielu plików podczas jednej sesji

`[Ctrl W] fraza`	szukaj frazy
	`[Ctrl+W]` szukaj kolejnego wystąpienia
	`[Alt+B]` tryb szukania wstecz
`[Ctrl W] [Ctrl T] {nr}`	skocz do linii nr.
`[Ctrl K]`	skasuj aktualną linię, ale zapamiętaj ją w schowku
`[Ctrl U]`	wstaw tu linię ze schowka
`[Alt Shift #]`	numeruj linie
`[Ctrl R]`	otwórz nowy plik w nowym „buforze”. (wymaga `nano -F`)
`[Alt > ]` `[Alt < ]`	przeskok do edycji następnego / poprzedniego pliku

`$ nano -v {plik}` : Tryb read-only (czytnik).
Ma koloryzację. Z drugiej strony, długie pliki wczytuje powoli.

⦿ Zdalne kopiowanie

- Kopia z/na zdalny komputer — `rsync` lub `scp`. rsync jest sprytniejszy.

💡 Propozycja: `rsync -avzP {źródło} {login@node:ścieżka/od/home}`

gdzie:

`a` (archive)	zachowa właściwości plików
`z` (zip)	kompresja w locie (szybciej)
`P` (partial+progress)	pokazuje postęp kopiowania + przy przerwaniu pozwala wznowić

`rsync` sprawdza, czy części paczki nie ma już w miejscu docelowym.
Kopiuje, gdy ich nie ma lub gdy plik docelowy jest starszy (uzysk czasu, przydatne do backupu)

- Np. utwórz jakiś plik: `$ touch abc.def`

i umów się z osobą obok na skopiowanie tego pliku (osoba musi wpisać hasło)

```
$ rsync -avzP abc.def LoginSasiada@tempac.okwf.fuw.edu.pl:
```

Ponieważ po znaku : nie ma nic, to plik osiądzie w katalogu głównym u tej osoby.

Umów się z osobą, aby skasowała jakiś podkatalog w Smash. Spróbuj wkopiować całość:

```
$ rsync -avzP Smash/* LoginSasiada@tempac.okwf.fuw.edu.pl:Smash/*
```

Ściąganie pliku z internetu

- Dobrą propozycją jest komenda: `wget {url}`

  Gdy plików w folderze jest więcej: `wget -r -np -nH {url/folder}`

  gdzie:

`r`	(recursive)	ściąga też podfoldery
`np`	(no-parent)	nie ściągaj plików ze ścieżek wyżej
`nH`	(no-host)	nie twórz ścieżki z nazwą głównej domeny

Wynik będzie *prawie* dobry. Bo jeśli zawartość do ściągnięcia jest na którejś podścieżce od korzenia domeny, to lokalnie zostaną utworzone wszystkie te podścieżki (choć będą puste).

Aby i tego uniknąć, policzmy liczbę N ścieżek między domeną a naszym miejscem i dodajmy opcję:

`--cut-dirs=N`

- Np. otwórz w przeglądarce: `www.fuw.edu.pl/~kpias/nwp/Smash/`

  Aby ściągnąć tylko config.yaml: `$ wget www.fuw.edu.pl/~kpias/nwp/Smash/config.yaml`

Aby jednak ściągnąć całą paczkę, trzeba napisać:

```
$ wget -r -np -nH --cut-dirs=2 www.fuw.edu.pl/~kpias/nwp/Smash/
```

⦿ Kompresja i archiwizacja plików

- Typowy kompresor plików — `gzip {pliki}` → `{pliki}.gz`

  i dekompresja: `gunzip {pliki}.gz` lub `gzip -d {pliki}.gz`

  Uwaga: `gzip` kompresuje plik po pliku, nie zajmuje się zbieraniem w paczkę.

  Np.:
```
$ cd run1/outdir
$ ls -ogh  particle_lists.oscar
$ gzip  particle_lists.oscar
$ ls -ogh  particle_lists.oscar.gz
$ gunzip   particle_lists.oscar.gz
$ cd ../../
```

- Pakowacz plików: `tar {archiwum} {plik1} {plik2} …` {również wildcards}

  Typowe pakowanie: `tar czvf {archiwum.tgz} {pliki źródłowe}`

  gdzie:

`c` (create)	utwórz archiwum
`z` (zip)	skompresuj
`f` (file)	dotyczy plików

  Wypis zawartości archiwum `tar tf {archiwum}`

  Typowe rozpakowywanie `tar xzvf {archiwum.tgz}`

  gdzie:

`x` (extract)	wypakuj

  Np.:
```
$ tar czvf Smash.tgz *
$ ls -ogh  Smash.tgz
$ tar tf   Smash.tgz
$ mkdir test ; mv Smash.tgz test ; cd test
$ tar xcvf Smash.tgz
$ ls -ogh
$ cd .. ; rm -r test
```

⦿ *Dodatek:* Przy Big Data :

- Wielowątkowy kompresor: `pigz {pliki}` lub `pigz -n {L. wątków} {pliki}`
- Połączenie tar i pigz `tar -c -I pigz -f archive.tgz {pliki_zrodlowe}`

⦿ Jak obsługiwać procesy.

- Podejrzyjmy je w monitorze: `$ top lub htop` (wyjście: q)
- i wypiszmy w terminalu: `$ ps`

(Skrót PID oznacza numer procesu).
Widzimy jednak tylko procesy w naszej sesji.
Aby zobaczyć wszystkie (nasze) procesy, wpiszmy:

`$ ps -u {login}`

Warto też spojrzeć na ich hierarchię: `$ ps -u {login} --forest`

Widzimy, że każdy proces jest „podpięty”
pod poprzednika, który go wywołał.

- Skasujmy proces, podając jego PID: `$ kill {PID}`

kasowanie na silniejszym priorytecie: `$ kill -9 {PID}`

Uwaga: skasowanie procesu-rodzica wyśle sygnał kasowania do procesów-dzieci.
Zatem, skasowanie terminala (lub sesji ssh) wyłączy wszystkie programy pod niego podpięte.

⦿ *Dodatek*

- Zamknięcie okna graficznego: `$ xkill` (a teraz kliknij myszką w okno)
- Wypiszmy „rodziców” procesów (PPID): `$ ps -f`
- Wypiszmy nr/y procesów, gdy znamy nazwę: `$ pidof {nazwa}`

⦿ Procesy c.d.: front, tło, pauzowanie

Włączmy w sesji program:

```
$ sleep 200
```

Tryb, w którym działa teraz proces, nazywa się foreground (front):
proces ma kontakt z terminalem, może tam kierować napisy.

Zapauzujmy teraz nasz program:

```
$ [Ctrl Z]
```

Wypiszmy listę zadań włączonych w naszej sesji:

```
$ jobs
[1]+  Stopped                 sleep 200
```

Nasze zadanie ma na tej liście nr. 1. Wznówmy je,
ale niech przejdzie do trybu background (tło):

```
$ bg %1
```

⇒ nie blokuje terminala, ale działa (sprawdź: `jobs`).

Wysuńmy je teraz z tła na front:

```
$ fg %1
```

Teraz przerwijmy nasz proces (zakończmy go):

```
[Ctrl C]
```

▸ Operator & (ampersand)

Włączmy program z oknem graficznym:

```
$ gedit
```

Włączenie grafiki blokuje terminal aż do jej wyłączenia (lub zapauzowania przez [Ctrl Z]).
Linux ma operator & (ampersand) , który od razu wpuszcza program do tła.

Spróbujmy:

```
$ gedit &
```

⇒ mamy kontakt i z terminalem, i z edytorem.

Nb.: procesy w tle są wciąż podpięte pod sesję:

```
$ ps -u {login} --forest
```

⦿ Pliki konfiguracyjne basha:

`~/.bashrc` wywoływany podczas logowania (ssh lub otwarcie desktopu)
`~/.bash_login` wywoływany np. przy nowym terminalu (w ramach tego samego logowania)

Przykładowy, prosty plik `~/.bash_login`

```
#!/bin/bash
echo Running ~/.bash_login
alias ls="ls --color"
alias fn="find . -name"
alias nano="nano -FP"
alias cgrep="grep --color -inr --include=*.{C,cpp,cxx,c,h,hpp}"
alias rsync="rsync -avzP"
```

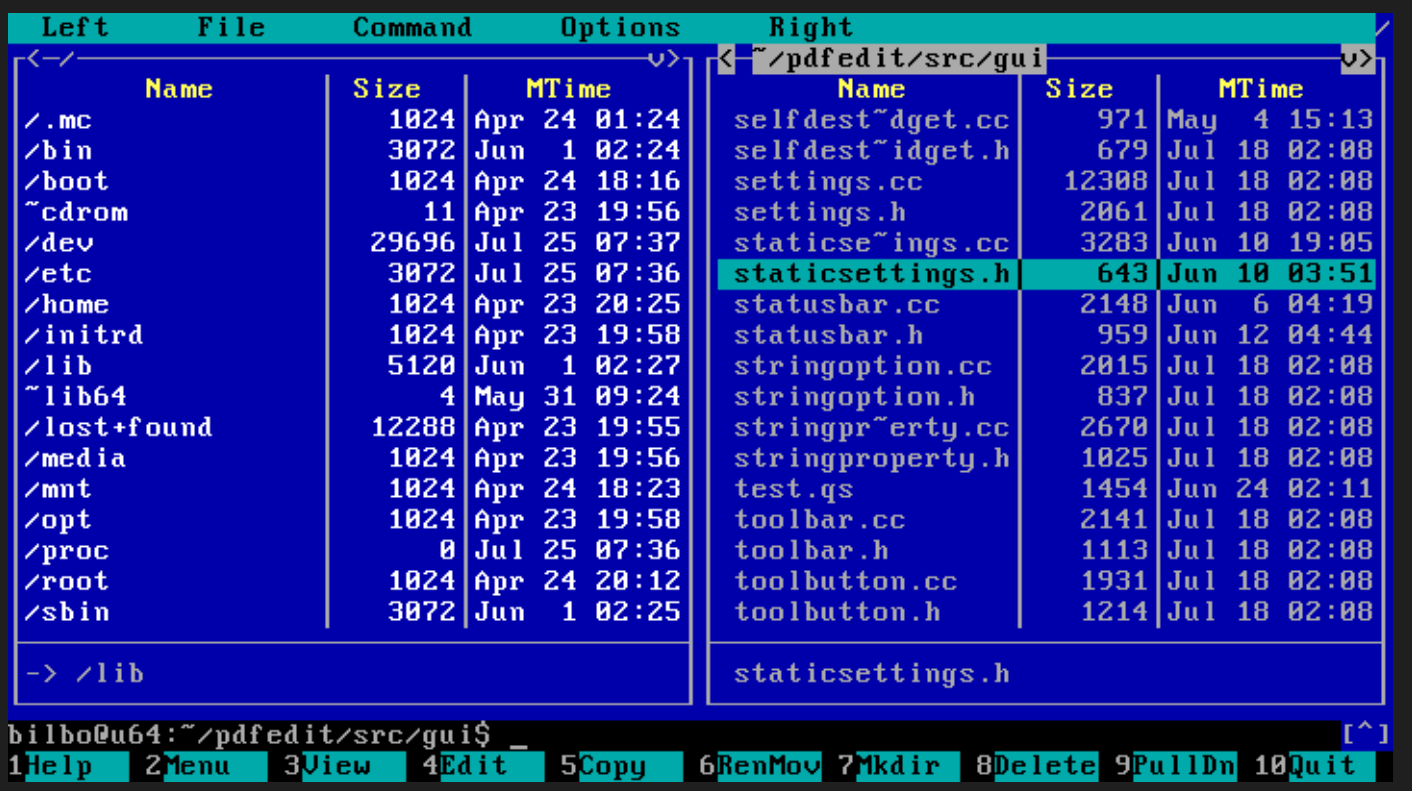
Nadajmy mu prawo do wykonania: `$ chmod 755 ~/.bash_login`

Dla pewności, że się uruchomi, do .bashrc dodajmy na koniec:

```
. ~/.bash_login
```

⊕ *Dodatek*:

- mc (Midnight Commander) : tekstowa przeglądarka plików



[Tab] Przeskok pomiędzy oknami
[F3] Czytnik pliku
[F8] Skasuj plik (ratuje, gdy plik ma w nazwie trudne znaki kontrolne)
[F10] Wyjście

Porównywanie treści plików:

Ściągnijmy dwie wersje tego samego pliku (wcześniejsza i późniejsza), nieco się różniące. Np:

```
$ mkdir diff ; cd diff
$ wget https://gitlab.gnome.org/GNOME/meld/-/raw/meld-3-12/setup.py?ref_type=heads -O setup.py.1
$ wget https://gitlab.gnome.org/GNOME/meld/-/raw/meld-3-22/setup.py?ref_type=heads -O setup.py.2
```

- Podstawową porównywarką jest diff. Napiszmy:

```
$ diff  setup.py.v1  setup.py.v2
```

i dostaniemy wypis różnic. Symbole: <,> = „w 1. , 2. pliku" . c = change , d = deletion , a = addition

```
1c1
< #!/usr/bin/env python
---
> #!/usr/bin/env python3
3d2
< from distutils.core import setup
4a4,9
> import pathlib
> from distutils.core import setup
...
```

- Znacznie czytelniejszy jest meld, ale nie wszędzie dostępny. Można edytować oba pliki!