



Politechnika
Wrocławska

Systemy operacyjne 2

Wykład

Niepełne wprowadzenie do zawłości Basha

Szymon Datko

szymon.datko@pwr.edu.pl

Wydział Informatyki i Telekomunikacji,
Politechnika Wrocławska

semestr letni 2022/2023



Cel tego wykładu

- ▶ Omówienie działania podstawowej powłoki systemu Linux.
- ▶ Ten wykład nie skupia się konkretnie na pisaniu skryptów.
- ▶ Zamiast tego:
 - materiał powinien ułatwić zrozumienie co w powłoce się dzieje,
 - wskazane zostaną pewne problematyczne/nieoczywiste elementy,
 - podana zostanie garść dobrych praktyk co do pracy z konsolą,
 - w zamierzeniu wykład ma utorować dalszą, samodzielną naukę.



Co to jest powłoka systemowa?

”

- In computing, a shell is a computer program which exposes an operating system's services to a human user or other program. In general, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI), depending on a computer's role (...). It is named a shell because it is the outermost layer around the operating system.
- Command-line shells require the user to be familiar with commands and their calling syntax, and to understand concepts about the shell-specific scripting language (for example, bash).
- Graphical shells place a low burden on beginning computer users, and are characterized as being easy to use. Since they also come with certain disadvantages, most GUI-enabled operating systems also provide CLI shells.

”

Emulator terminala

”

- A terminal emulator, (...), is a computer program that emulates a video terminal within some other display architecture. Though typically synonymous with a shell or text terminal, the term terminal covers all remote terminals, including graphical interfaces. A terminal emulator inside a graphical user interface is often called a terminal window.
- A terminal window allows the user access to a text terminal and all its applications such as command-line interfaces (CLI) and text user interface (TUI) applications. These may be running either on the same machine or on a different one via telnet, ssh, or dial-up. On Unix-like operating systems, it is common to have one or more terminal windows connected to the local machine.
- Terminals usually support a set of escape sequences for controlling color, cursor position, etc. Examples include the family of terminal control sequence standards known as ECMA-48, ANSI X3.64 or ISO/IEC 6429.

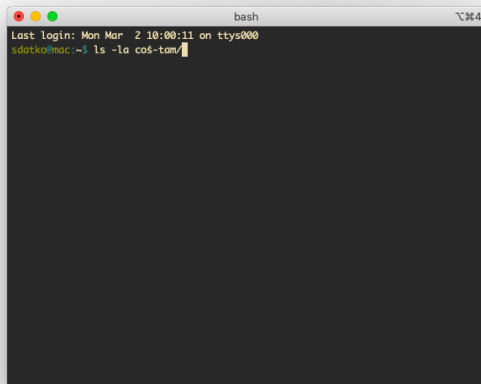
”

Relikt przeszłości?



Zdjęcie własne – 29.03.2022, piwnica budynku C-3, Politechnika Wroclawska.

Gdzie tu jest Bash?



```
bash
Last login: Mon Mar 2 10:00:11 on ttys000
sdatk@mac:~$ ls -la coś-tam/
```

Jaką część, z tego co widzimy na ekranie, stanowi faktycznie powłoka?
Co dokładnie się stanie po naciśnięciu klawisza [ENTER]?

Komenda: `ls -la coś-tam/` (1/2)

- W wyniku prezentowana jest zawartość katalogu o nazwie `coś-tam`.
- O ile ten katalog istnieje i mamy do niego odpowiednie prawa dostępu.
- A ile ma to wspólnego z Bashem?

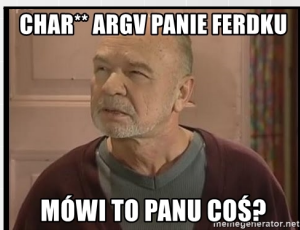
Komenda: `ls -la coś-tam/` (2/2)

- W wyniku prezentowana jest zawartość katalogu o nazwie `coś-tam`.
- O ile ten katalog istnieje i mamy do niego odpowiednie prawa dostępu.
- A ile ma to wspólnego z Bashem?
 - ▶ Zasadniczo nic.
 - ▶ Proces powłoki uruchomił proces potomny i zmienił program.
 - ▶ W sensie języka C: najpierw `fork()`, później `exec()`.
 - ▶ Proces rodzica czeka, zbiera wynik i kontynuuje pracę.
 - ▶ Za całą akcję (dla obserwatora) odpowiada program `ls`.



Program ls?

- Tak! Można użyć programu `whereis` do sprawdzenia tego.
- Gdzieś w systemie istnieje odpowiedni program do uruchomienia.
 - ▶ Zazwyczaj będzie to `/bin/ls` albo `/usr/bin/ls`.
- A co z tymi wszystkimi parametrami?
 - ▶ One są właśnie obsługiwane przez program `ls`.
 - ▶ Skąd żądany program o nich wie?
 - ▶ `int main(int argc, char** argv) { ... }`



Słowo na temat przełączników opcji

- Zasadniczo wyróżnia się opcje:
 - ▶ krótkie: pojedynczy minus i litera, np. `-l`;
 - ▶ długie: dwa minusy i napis, np. `--human-readable`.
- Technicznie tam są minusy (-), a nie myślniki (--), naprawdę :-)
 - ▶ Przeszkadza to w wyszukiwaniu odpowiednich opcji w Google...
- Czasem po spacji lub znaku = można określić parametry do opcji,
 - ▶ na przykład: `--sort=size`;
 - ▶ ewentualnie: `--sort size`.
- Opcje krótkie zazwyczaj można grupować `-l -a = -la`.
- **Nie zawsze musi tak być!**
 - ▶ Jest to jedynie konwencja i nie wszystkie programy ją stosują.
 - ▶ W tym niektóre powszechnie używane: `find . -iname '*.jpg'`.
 - ▶ Hasło do wyszukania do poduszki: `getopt`.

Ile jest komend w Bashu?

- Czy chwilę temu powiedziane było **program** `whereis`?

- ▶ Otóż to! → `/usr/bin/whereis`

- Kilka innych przykładów:

- ▶ `mv` → `/usr/bin/mv`

- ▶ `cp` → `/usr/bin/cp`

- ▶ `true` → `/usr/bin>true`

- ▶ `if` → `???` # brak wyniku

- ▶ `cośtam` → `???` # brak wyniku



- Kiedy wpisano nazwę jakiegoś programu, zostaje on uruchomiony.

- Skąd powłoka wie, co ma uruchomić?

- ▶ Katalogi do przeszukania są określone w zmiennej `$PATH`.

- ▶ Ścieżki są sprawdzane w kolejności w jakiej je podano.

Zmienne powłoki

- Jest to każda zmienna zadeklarowana w powłoce Bash.
- Przykład deklaracji:
 - ▶ `nazwa=wartość`
 - ▶ Dookoła znaku = **nie może być** żadnej spacji! (**SUPER WAŻNE**)
- Stosowane głównie, żeby sparametryzować wywołanie jakiejś komendy,
 - ▶ na przykład: `ls -la "${nazwa}";`
 - ▶ szczególnie, kiedy jakaś wartość powtarza się wielokrotnie w kodzie.
- Mogą zostać przekazane do procesu potomnego → zmienne środowiskowe.
 - ▶ Aby do tego doszło, zmienna musi zostać wyeksportowana.
 - `nazwa=wartość # najpierw deklaracja`
 - `export nazwa # lub: export nazwa=wartość`
 - `komenda # $zmienna będzie tzw. zm. środowiskową`
 - ▶ Można też zdefiniować zmienną przy wywoływaniu komendy.
 - `nazwa=wartość komenda # analog. do tego wyżej`
 - ▶ W macOS: wyjątek stanowi zmienna `$PATH`, czytana z `/etc/paths`.

Zmienne środowiskowe

- Kolejny zestaw wartości (opcji), przekazywanych do procesu.
- Należy zawsze uważać, zmieniając wartość tych zmiennych!
- Bieżące zmienne można wyświetlić za pomocą komendy `env`.
 - ▶ Można je znaleźć w systemie plików, tak jak inne ciekawe rzeczy.
 - ▶ Ścieżka do przejrzania: `/proc/${numer_procesu}/`.
 - ▶ Przykłady ciekawych rzeczy:
 - `environ` – zmienne środowiskowe,
 - `exe` – dowiązanie do pliku wykonywanego programu,
 - `cwd` – dowiązanie do katalogu, w którym działa proces.
- W jaki sposób program napisany w języku C je otrzymuje?
 - ▶ Używając funkcji `getenv()` z biblioteki `stdlib.h`.
 - ▶ `int main(int argc, char** argv, char** envp) { ... }`
~~~~~

# Uwagi dotyczące używania zmiennych

- Kiedy zmienna została zdefiniowana, można jej użyć w komendach.
  - ▶ **Uwaga!** Domyślnie Bash nie krzyczy, jeśli zmienna nie istnieje!
  - ▶ Można to specjalnie włączyć: `set -o nounset`.
- Kilka wariantów odwołania się do wartości:
  - ▶ po prostu, np. `echo $nazwa # czy to może źle działać?`,
  - ▶ lepiej, np. `echo "$nazwa" # rozważmy: ls $katalog`,
  - ▶ najlepiej, np. `echo "${nazwa}"`.
- Ten ostatni wariant pozwala na dodatkowe manipulacje zawartością.
  - ▶ Hasło do wyszukania do poduszki: Bash Shell Parameter Expansion.
  - ▶ Poza tym przestrzeganie spójności w kodzie jest bardzo ważne.
- Obecnie powłoki wspierają też złożone typy, jak na przykład **tablice**.

# Przykład działania zmiennych środowiskowych (1/2)

```
[sdatko@polluks ~]$ ls MojeDokumenty
ls: nie ma dostępu do 'MojeDokumenty': Nie ma takiego pliku ani katalogu
[sdatko@polluks ~]$ LC_ALL=en_US.UTF-8 ls MojeDokumenty
ls: cannot access 'MojeDokumenty': No such file or directory
[sdatko@polluks ~]$ LC_ALL=fr_FR.UTF-8 ls MojeDokumenty
ls: impossible d'accéder à 'MojeDokumenty': Aucun fichier ou dossier de ce type
[sdatko@polluks ~]$ LC_ALL=de_DE.UTF-8 ls MojeDokumenty
ls: Zugriff auf 'MojeDokumenty' nicht möglich: Datei oder Verzeichnis nicht gefunden

[sdatko@polluks ~]$ PS1="[über@BASH]$ " LC_ALL=de_DE.UTF-8 bash --noprofile --norc
[über@BASH]$ datum "+%A, %d %B"
bash: datum: Kommando nicht gefunden.
[über@BASH]$ alias Datum=date
[über@BASH]$ alias Kopf=head
[über@BASH]$ Datum "+%A, %d %B"
Freitag, 17 April

[über@BASH]$ gcc keine-file
gcc: Fehler: keine-file: Datei oder Verzeichnis nicht gefunden
gcc: schwerwiegender Fehler: keine Eingabedateien
Kompilierung beendet.

[über@BASH]$ vim --help | Kopf -n 5
VIM - Vi IMproved 8.2 (2019 Dec 12 kompiliert am Feb 09 2021 23:51:55)
```

```
Verwendung: vim [Argumente] [Datei ..]      editiere die angegebenen Datei(-en)
oder: vim [Argumente] -                    lese Text von stdin
oder: vim [Argumente] -t tag                öffne Datei in der der Tag definiert wurde
oder: vim [Argumente] -q [Fehler-Datei]    öffne Datei mit erstem Fehler
```



# Przykład działania zmiennych środowiskowych (2/2)

```
[sdatko@polluks ~]$ LC_ALL=ja_JP.UTF-8 gcc nieistniejący-plik
```

```
/usr/bin/ld: nieistniejący-plik が見つかりません: そのようなファイルやディレクトリはありません
collect2: エラー: ld はステータス 1 で終了しました
```

```
[sdatko@polluks ~]$ LC_ALL=ja_JP.UTF-8 vim --help | head -n 21
```

```
VIM - Vi IMproved 9.0 (2022 Jun 28, compiled Feb 11 2023 18:31:20)
```

使用法: vim [引] [ファイル...] 指定されたファイルを編集する  
 もしくは: vim [引] - 標準入力からテキストをむ  
 もしくは: vim [引] -t タグ タグが定義されたところから編集する  
 もしくは: vim [引] -q [errorfile] 最初のエラーで編集する

引:

|                |                             |
|----------------|-----------------------------|
| --             | このあとにはファイル名だけ               |
| -g             | GUIで起動する ("gvim" と同じ)       |
| -f or --nofork | フォアグラウンド: GUIを始めるときにforkしない |
| -v             | Viモード ("vi" と同じ)            |
| -e             | Exモード ("ex" と同じ)            |
| -E             | 改良Exモード                     |
| -s             | サイレント(バッチ)モード ("ex" 用)      |
| -d             | 差分モード ("vidiff" と同じ)        |
| -y             | イージーモード ("evim" と同じ、モード無)   |
| -R             | 用モード ("view" と同じ)           |
| -Z             | 制限モード ("rvim" と同じ)          |
| -m             | 更 (ファイル保存時) をできないようにする      |
| -M             | テキストの編集を行なえないようにする          |



## Kiedy białe znaki nie mają znaczenia?

- Koncepcja podobna, jak na przykład w języku C,
- Wszystkie poniższe wywołania dają taki sam wynik:
  - ▶ `ls -l -a MojeDokumenty/`,
  - ▶ `ls -l -a MojeDokumenty/`,
  - ▶ `ls -l -a MojeDokumenty/`.
  - ▶ `ls -la MojeDokumenty/`.
- Ostatni przedstawiony wyżej przykład stanowi pewnego rodzaju wyjątek.
  - ▶ Trzy pierwsze: identyczne wywołanie komendy `ls` z trzema opcjami.
  - ▶ Ostatni to wywołanie komendy `ls` z dwiema opcjami.
- Domyślnie białe znaki separują argumenty w powłoce.
  - ▶ Konfiguruje to zmienna środowiskowa `$IFS` (domyślnie `'\t\n'`).
- Liczba separatorów w praktyce nie ma znaczenia.
- To wszystko jest prawdą, ale...

## Kiedy białe znaki mają duże znaczenie?

- Nie można ich stosować przy deklaracjach zmiennych.
- Nie można ich pomijać przy rozdzielaniu argumentów.
- W przeciwnym wypadku istotnie może się zmienić sens komendy:

▶ `ls -l -a MojeDokumenty/`

# Uruchom program `ls` z trzema argumentami.

▶ `ls-l -aMojeDokumenty/`

# Uruchom program `ls-l` z jednym argumentem.

▶ `zmienna = jakaś wartość`

# Uruchom program `zmienna` z trzema argumentami.

▶ `zmienna= jakaś wartość`

# Uruchom program `jakaś` z pustą zmienną środowiskową `$zmienna` i argumentem `wartość`.

▶ `zmienna=jakaś wartość`

# Uruchom program `wartość` z dodatkową zmienną środowiskową `$zmienna`.

▶ `zmienna=jakaś\ wartość`

# Utwórz zmienną `$zmienna` o wartości `jakaś wartość`.

- Znak nowej linii ma specjalne znaczenie – kończy komendę.

# Słowo na temat łańcuchów znaków

- Zasadniczo dostępne są dwa warianty:
  - ▶ 'wykorzystujące apostrofy',
  - ▶ "używające cudzysłowów".
- Pozwalają na zawieranie spacji i znaków specjalnych jako argumenty.
- Różnią się tym, czy wstawiona zostaje zawartość zmiennej.
  - ▶ `echo ~ $osoba ma kota,`  
# Program `echo` otrzyma 3 lub więcej argumentów, zależnie od wartości `$osoba`.
  - ▶ `echo '~ $osoba ma kota',`  
# Program `echo` otrzyma 1 argument, jednym z nich będzie napis `$osoba`.
  - ▶ `echo "~ $osoba ma kota",`  
# Program `echo` otrzyma 1 argument, zamiast `$osoba` zostanie wstawiona wartość.
- Alternatywnie można po prostu użyć znaku modyfikatora/ucieczki.
  - ▶ Tak formalnie nazywa się wtyłciach `\ ;-)`
  - ▶ Przykład: `napis=To\ jest\ jakaś\ wartość\ ze\ spacjami.`

## Znaki specjalne w komendach

- Domyślnie komenda kończy się z chwilą napotkania końca linii.
- Nie dzieje się tak, jeśli wstawiono na końcu wtyłciach \ ;-)
- Podobnie można sprawić, że spacja nie będzie rozdzielać argumentów.
- Kilka przykładowych znaków specjalnych:
  - ▶ # – rozpoczyna komentarz,
  - ▶ ; – zakończenie bieżącej komendy i uruchomienie jej,
  - ▶ & – kończy komendę i uruchamia ją w tle (bez `wait`, patrz slajd 9),
  - ▶ ~ – ścieżka bezwzględna do katalogu domowego,
  - ▶ ( ) [ ] { } – wiele znaczeń, zależnych od miejsca użycia...
- Dopasowywanie ścieżek:
  - ▶ \* – dowolny pasujący ciąg znaków,
  - ▶ ? – dokładnie jeden jakikolwiek znak,
  - ▶ [abc] – jeden z ze znaków a, b lub c,
  - ▶ [0-7] – jeden znak z zakresu 0-7,
  - ▶ hasło do wyszukania do poduszki: `glob` (programming).

# Instrukcje warunkowe

- Modelowa składnia wygląda następująco:

```
if komenda1; then
    komenda2
elif komenda3; then
    komenda4
else
    komenda5
fi
```

- Przykład z życia:

```
if [ ! -d MojeDokumenty/ ]; then
    mkdir MojeDokumenty/
fi
```

- Dlaczego i jak to działa?

- ▶ Wykorzystywany jest mechanizm kodów powrotu (ang. *exit status*).
- ▶ Początkowo używany był program `test`, który nazywa się też `[`.
- ▶ Dziś Bash implementuje własny, wbudowany wariant pod nazwą `[`.
- ▶ Bash ma także swój inny wariant, o większych możliwościach: `[[`.

# Kody powrotu z programów

- Wartość numeryczna, dostępna w zmiennej `$?` po wykonaniu komendy.
- Mechanizm pozwalający na rozpoznanie wyniku działania programu.
  - ▶ Jeśli program zakończył się bez błędu, to wartość `0`.
  - ▶ Jeśli pojawił się błąd, to wartość jest niezerowa.
  - ▶ Dokładna wartość może wskazywać przyczynę błędów.
- Ponownie, jest to wyłącznie **konwencja** – implementuje ją program.
  - ▶ Pozwala wygodnie użyć niektórych komend jako warunku akcji.
  - ▶ Przykład z życia: poprzedni slajd – gdy używamy programu `test`.
  - ▶ Czasem trzeba mimo wszystko przejrzeć całe wyjście z komendy.
  - ▶ Przykład z życia: aplikacja zawsze odpowiada statusem HTTP 200...
- Skąd biorą się te wartości?
  - ▶ W języku C: wartość zwracana przez funkcję `main()`.
    - Inny sposób: wywołanie funkcji `exit()` z argumentem...

# Pętle

- Najczęściej stosowana jest pętla `for` podobna do tej z języka Python.
- Używana jest ona do przejrzenia jakiejś kolekcji argumentów/wartości.
- Modelowa składnia wygląda następująco:

```
for nazwa in argument1 argument2 'argument 3' ...; do
    komenda
done
```

- Przykład z życia:

```
for numer in $(seq 0 9); do
    touch "plik${numer}.txt"
done
for nazwa in plik*; do
    echo "${nazwa}"
done
```

- Dlaczego i jak to działa?

- ▶ W pierwszym przykładzie użyto przechwyconego wyniku programu `seq`.
- ▶ Każde kolejne słowo trafi do zmiennej `$numer` w kolejnych iteracjach pętli.
- ▶ W drugim przykładzie, `*` jest znakiem specjalnym powłoki.<sup>1</sup> <sup>1</sup>(hasło: glob)
- ▶ Fraza `plik*` zostanie zastąpiona przez pasujące nazwy plików.
- ▶ Czyli pętla `for` będzie wykonana dla argumentów `plik0`, `plik1`, `plik2` itd.

- Istnieje także pętla `while` oraz wariant `for` podobny do tego z języka C.



# Skrypty powłoki

- Prawie jak zwykłe programy binarne (mogą być wykonywalne).
- Po prostu zbiór pojedynczych komend do uruchomienia w kolejności.
- Uruchamianie:
  - ▶ wariant podstawowy: `bash skrypt.sh`,
  - ▶ odpluskwanie: `bash -x skrypt.sh` (opcja `-x = set -o xtrace`),
  - ▶ `./skrypt.sh` (gdy plik jest wykonywalny, używany jest shebang).
- shebang – specjalna instrukcja jak uruchomić dany plik tekstowy.
  - ▶ Jest to pierwsza linijka skryptu, np. `#!/bin/bash` (tak, jest to komentarz!).
  - ▶ Po znakach `#!` wskazuje się ścieżkę do pożądanego interpretera.
  - ▶ Położenie uruchamianego pliku to argument dla interpretera.
  - ▶ Obecnie często jest/powinien być to `#!/usr/bin/env bash`.
  - ▶ Do przemyślenia – co się stanie, kiedy użyjemy `#!/bin/rm?`

# shellcheck

- Niezwykle przydatne narzędzie podczas codziennych potyczek.
- Zgłasza zarówno realne, jak i potencjalne problemy ze skryptami.
- Użycie z poziomu konsoli:

▶ `shellcheck skrypt.sh`

- Przykład działania:

In skrypt.sh line 2:

```
katalog='jakaś nazwa'
```

```
^-----^ SC2034: katalog appears unused. Verify use (or export if used externally).
```

In skrypt.sh line 3:

```
mkdir -p $katlog
```

```
^-----^ SC2154: katlog is referenced but not assigned (did you mean 'katalog'?).
```

```
^-----^ SC2086: Double quote to prevent globbing and word splitting.
```

Did you mean:

```
mkdir -p "$katlog"
```

For more information:

<https://www.shellcheck.net/wiki/SC2034> -- katalog appears unused. Verify us...

<https://www.shellcheck.net/wiki/SC2154> -- katlog is referenced but not assi...

<https://www.shellcheck.net/wiki/SC2086> -- Double quote to prevent globbing ...

# Jak się uczyć bez obaw?

- Część komend, niepoprawnie użyta, może skutkować utratą danych.
  - ▶ W szczególności chodzi tutaj o program `rm`.
  - ▶ Także programy `mv` i `cp`, jeśli użyć ich z nieroztropnie.
    - Programy `cp`, `mv` i `rm` mają przydatną opcję `-i`.
  - ▶ Zwykły zapis do plików (`>` i `>>`) to również ryzyko utraty danych.
    - `set -o noclobber` pozwala się częściowo zabezpieczyć.
- Naukę lepiej przeprowadzić w środowisku bez ważnych dokumentów.
  - ▶ Nie, osobny katalog na dysku może nie wystarczyć!
  - ▶ Maszyna wirtualna sprawdzi się świetnie.
  - ▶ Można też użyć technologii kontenerowych (np. Docker, podman).
- Poza tym **zawsze** sprawdzaj w którym terminalu pracujesz.
- Zawsze też pilnuj zawartości schowka i nie wklejaj w terminal bez uwagi!

## Najważniejsze porady i rzeczy do zapamiętania

- Bash jest interfejsem do uruchamiania programów i sterowania nimi.
- Da się go lubić, a nawet kochać, jeśli pamięta się o paru drobiazgach.
- Google/Stack/... zawsze spoko, ale warto się oswoić z programem `man`.
- W skryptach dobrze jest stosować długie nazwy opcji dla czytelności.
- Zawsze kontroluj co zawiera zmienna i uwzględniaj białe znaki.
- Nigdy nie zmieniaj wartości zmiennych `$PATH` i `$IFS`.
- Opcja `xtrace` oraz program `shellcheck` to dobrzy przyjaciele.



To wszystko na dziś.

# Pytania?

To wszystko na dziś.

~~Pytania?~~ Dziękuję za uwagę!