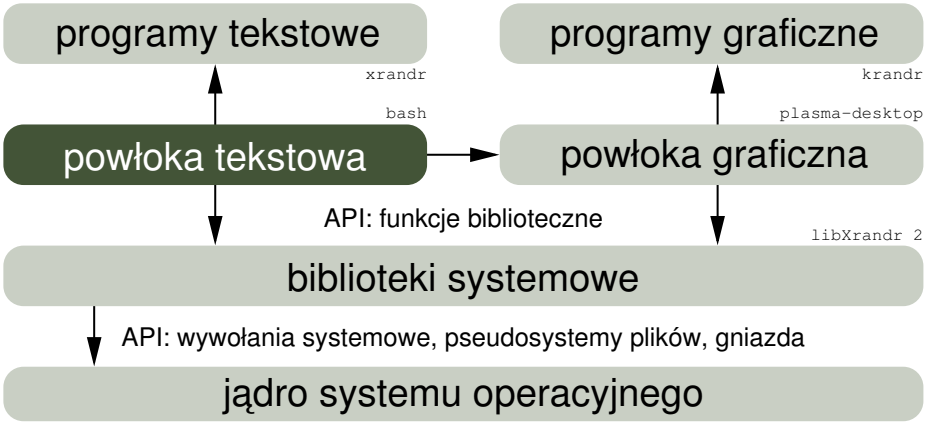
Podstawowy warsztat informatyka

Zajęcia nr 6: Powłoka systemowa

Gościnnie TWI

13 listopada 2018

Powłoka tekstowa w systemach uniksowych



Programy tekstowe i graficzne. Przykład: xrandr i krandr

Program tekstowy

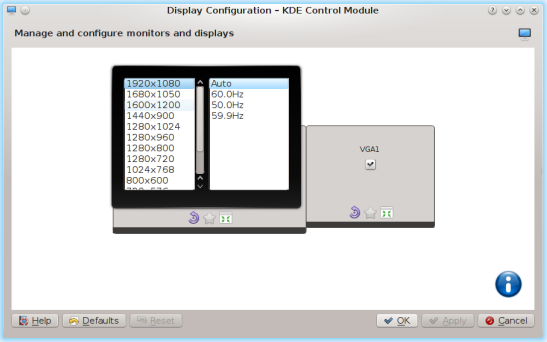
```
... 1-$ scrot  2*$ man  8@$ msg  ...                Mon Nov 12 18:38 BAT100% SCR30%
XRANDR(1)                  General Commands Manual                  XRANDR(1)

NAME
       xrandr - primitive command line interface to RandR extension

SYNOPSIS
       xrandr  [--help]  [--display display] [-q] [-v] [--verbose] [--dryrun]
       [--screen snum] [--q1] [--q12] [--current] [--noprimary] [--panning
       widthxheight[+x+y[/track_widthxtrack_height+track_x+track_y[/bor-
       der_left/border_top/border_right/border_bottom]]]] [--scale xxy]
       [--scale-from wxh] [--transform a,b,c,d,e,f,g,h,i] [--primary] [--prop]
       [--fb widthxheight] [--fbmm widthxheight] [--dpi dpi] [--newmode name
       mode] [--rmmode name] [--addmode output name] [--delmode output name]
       [--output output] [--auto] [--mode mode] [--preferred] [--pos xxy]
       [--rate rate] [--reflect reflection] [--rotate orientation] [--left-of
       output] [--right-of output] [--above output] [--below output] [--same-
       as output] [--set property value] [--off] [--crtc crtc] [--gamma
       red:green:blue] [--brightness brightness] [--o orientation] [-s size]
       [-r rate] [-x] [-y] [--listproviders] [--setprovideroutputsource
       provider source] [--setprovideroffloadsink provider sink]
Manual page xrandr(1) line 1 (press h for help or q to quit)
```

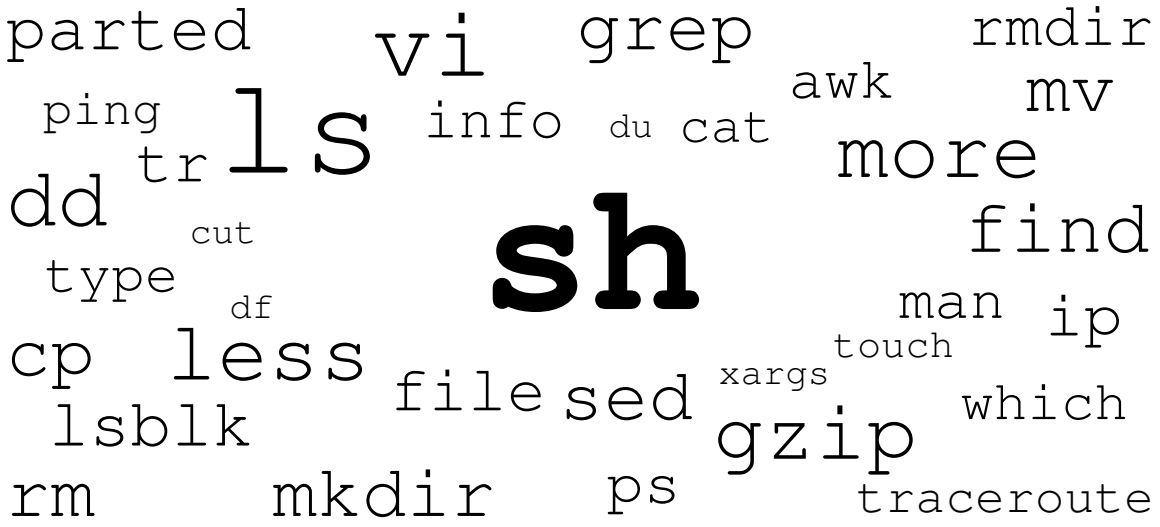
- Udostępnia wszelkie zaimplementowane usługi.
- Duża niezawodność i powtarzalność działania.
- Duże możliwości diagnostyki w razie problemów.
- Łatwa automatyzacja (skryptowanie).
- Wymaga przeczytania podręcznika obsługi.

Program graficzny

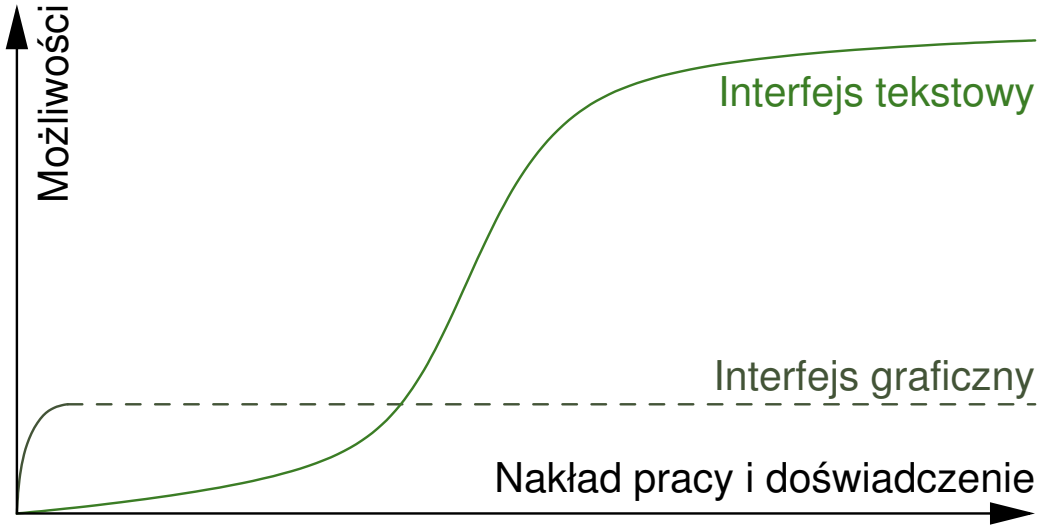


- Udostępnia niewielki podzbiór usług.
- Zawodność i niepowtarzalność działania.
- Niewielkie możliwości diagnostyki.
- Brak możliwości automatyzacji.
- Intuicyjny interfejs, bez potrzeby nauki.

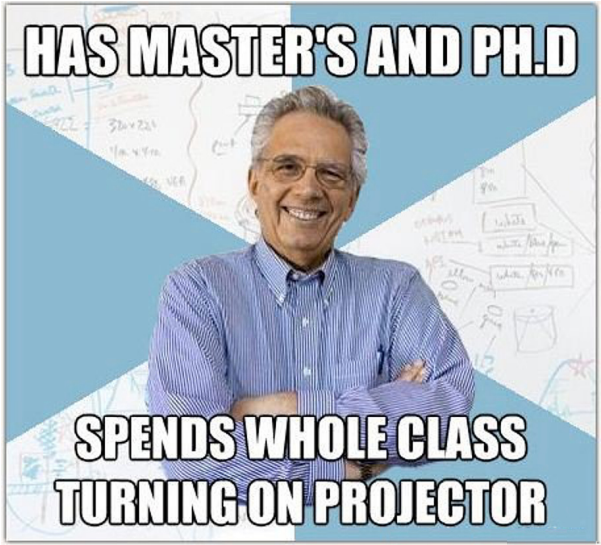
Filozofia Uniksa



Nakład pracy i doświadczenie



Konfiguracja monitora w praktyce



Żartowałem!

```
xrandr --fb 1366x768 --output eDP-1 --pos 0x0
   --output HDMI-2 --reflect x

xrandr --fb 1366x768 --output eDP-1 --pos 0x0
   --output HDMI-2 --reflect xy

xrandr --fb 1366x768 --output eDP-1 --pos 0x0
   --output HDMI-2 --transform .3,-.3,300,.3,.3,-50,0,0,1

xrandr --fb 1366x768 --output eDP-1 --pos 0x0
   --output HDMI-2 --mode 1920x1080 --scale-from 1366x768
```

Powłoki w Linuksie

sh i kompatybilne

- sh (Bourne 1979)
- ash (Almquist)
- **bash** (Bourne Again)
- dash (Debian Almquist)
- ksh (Korn)
- mksh (MirBSD Korn)
- zsh (Zhong Shao)
- busybox

csh i kompatybilne

- csh (Berkeley C)
- tcsh (Tenex C)

Inne, egzotyczne

- yash (Yet Another)
- scsh (Scheme)

Dokumentacja GNU Bash

Dwa podobne, ale różne dokumenty:

- `bash(1)` (*Linux man page*, źródło: `groff`),
- *Bash Reference Manual*, Chet Ramey (CWRU), Brian Fox (FSF) (GNU Texinfo, źródło: `texi`).

Slang: „man bash" — długi i niezrozumiały dokument. Nic bardziej mylnego!

Warto też czytać podręczniki, np.:

- *bash Cookbook*, Carl Albing, JP Vossen, Cameron Newham, O'Reilly 2007. Pol. tłum.: *bash. Receptury*, Helion 2008.
- *Learning the bash Shell*, Cameron Newham, Bill Rosenblatt, O'Reilly, 3rd ed. 2005.
- *Bash Pocket Reference*, Arnold Robbins, O'Reilly, 2nd ed. 2016.

. . . i wiele innych.

man bash

BASH(1) General Commands Manual BASH(1)

NAME

bash − GNU Bourne-Again SHell

SYNOPSIS

bash [options] [command_string | file]

COPYRIGHT

DESCRIPTION

Bash is an **sh**-compatible command language interpreter that executes commands read from the standard input or from a file. **Bash** also incorporates useful features from the *Korn* and *C* shells (**ksh** and **csh**).

Bash is intended to be a conformant implementation of the Shell and Utilities portion of the IEEE POSIX specification (IEEE Standard 1003.1). **Bash** can be configured to be POSIX-conformant by default.

OPTIONS

All of the single-character shell options documented in the description of the **set** builtin command can be used as options when the shell is invoked. In addition, **bash** interprets the following options when it is invoked:

−c If the **−c** option is present, then commands are read from the first non-option argument *command_string*. If there are arguments after the *command_string*, they are assigned to the positional parameters, starting with **$0**.

1 Introduction

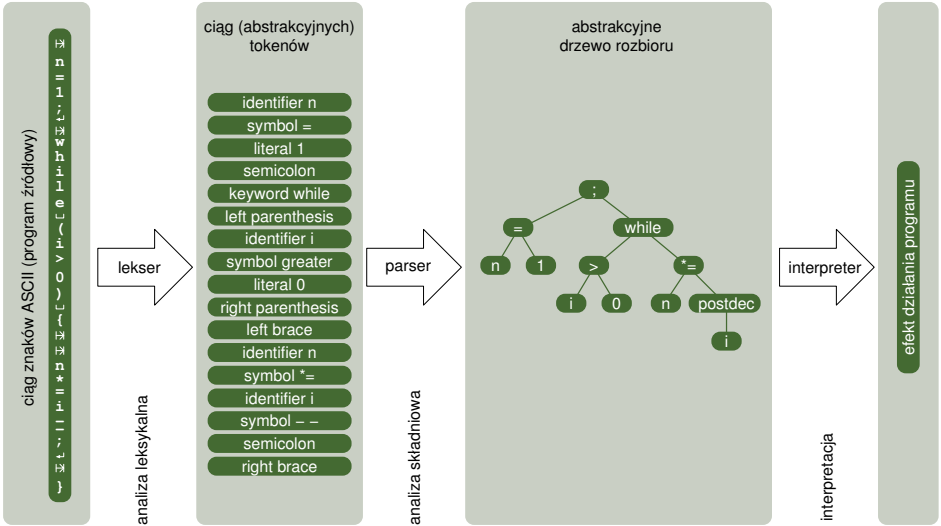
1.1 What is Bash?

Bash is the shell, or command language interpreter, for the GNU operating system. The name is an acronym for the '`Bourne-Again SHell`', a pun on Stephen Bourne, the author of the direct ancestor of the current Unix shell `sh`, which appeared in the Seventh Edition Bell Labs Research version of Unix.

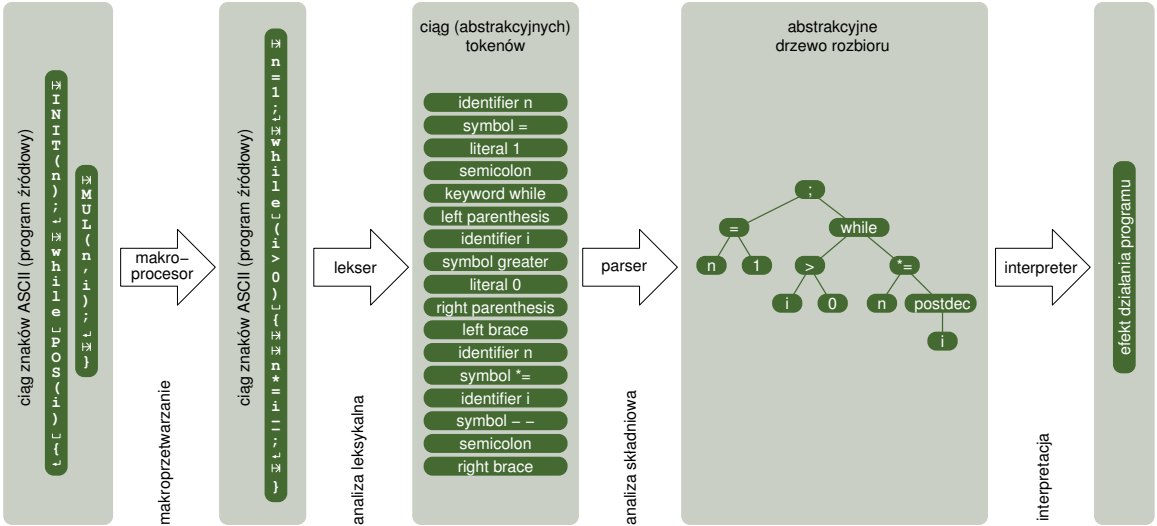
Bash is largely compatible with `sh` and incorporates useful features from the Korn shell `ksh` and the C shell `csh`. It is intended to be a conformant implementation of the IEEE POSIX Shell and Tools portion of the IEEE POSIX specification (IEEE Standard 1003.1). It offers functional improvements over `sh` for both interactive and programming use.

While the GNU operating system provides other shells, including a version of `csh`, Bash is the default shell. Like other GNU software, Bash is quite portable. It currently runs on nearly every version of Unix and a few other operating systems − independently-supported ports exist for MS-DOS, OS/2, and Windows platforms.

Dygresja: Kompilacja i wykonanie języków programowania



Kompilacja i wykonanie języków programowania z makroprocesorem



Preprocesor języka C (nie róbcie tego w domu)

```
#define if if (
#define then ) {
#define else } else {
#define fi }
#define begin {
#define end }

int main(int argc, char* argv[])
begin
  if argc > 1
    then
      printf("Hello %s!\n", argv[1]);
    else
      printf("Hello!\n");
  fi
  return 0;
end
```

cpp -P →

```
int main(int argc, char* argv[])
{
  if ( argc > 1
    ) {
      printf("Hello %s!\n", argv[1]);
    } else {
      printf("Hello!\n");
  }
  return 0;
}
```

Uwaga: bash nie jest zwykłym językiem programowania!

Podstawowe zadanie: uruchamianie programów

1. Wczytaj z linii poleceń nazwę programu i argumenty wywołania.
2. Zażądaj od jądra uruchomienia programu z podanymi argumentami.
3. Odbierz od jądra kod powrotu.
4. Powróć do punktu 1.

Dodatkowe możliwości

- Udogodnienia w przygotowywaniu poleceń do wykonania:
  - Mechanizmy makropodstawień.
  - Wyszukiwanie plików w wielu katalogach (`PATH`, `MANPATH` itp.).
- Prosty interpretowany język programowania pozwalający na podejmowanie decyzji warunkowych i iterację.

Działanie interpretera basha

Wykonuje w pętli poniższe czynności aż osiągnie koniec strumienia wejściowego, wykona instrukcję `exit`, natrafi na błąd itp.

Parsowanie

- Czyta tekst z konsoli lub pliku.
- Dzieli ciągi znaków na tokeny.
- Parsuje ciągi tokenów dzieląc je na instrukcje proste i złożone.
- Zatrzymuje się po przeczytaniu jednej kompletnej instrukcji.

Interpretacja

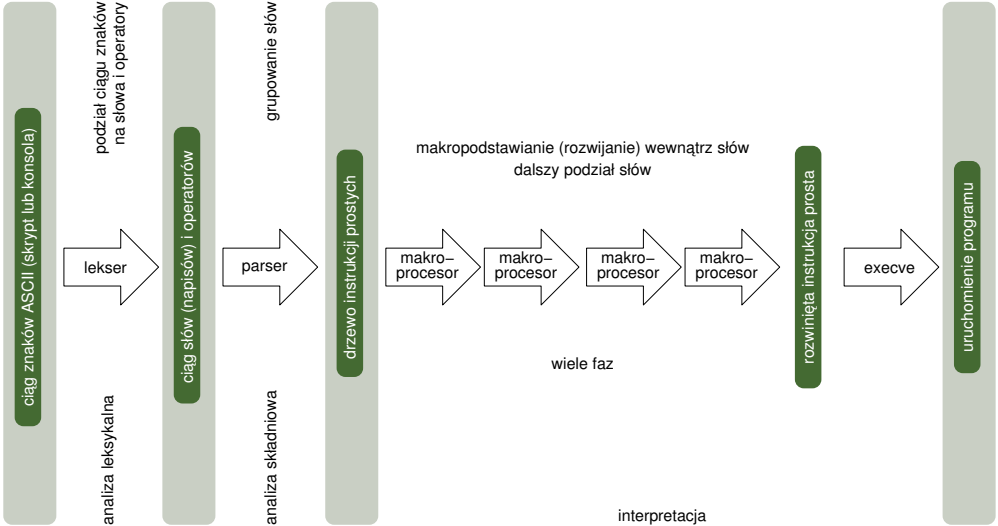
- Wykonuje cyklicznie poniższe czynności.
- Zgodnie z logiką programu wybiera kolejną instrukcję do wykonania.
- Dokonuje w niej szeregu *rozwinięć*.
- Wybiera i interpretuje umieszczone w niej przekierowania.
- Wykonuje instrukcję.
- Opcjonalnie czeka na jej zakończenie i odczytuje jej kod zakończenia.

Makropodstawienia w bashu

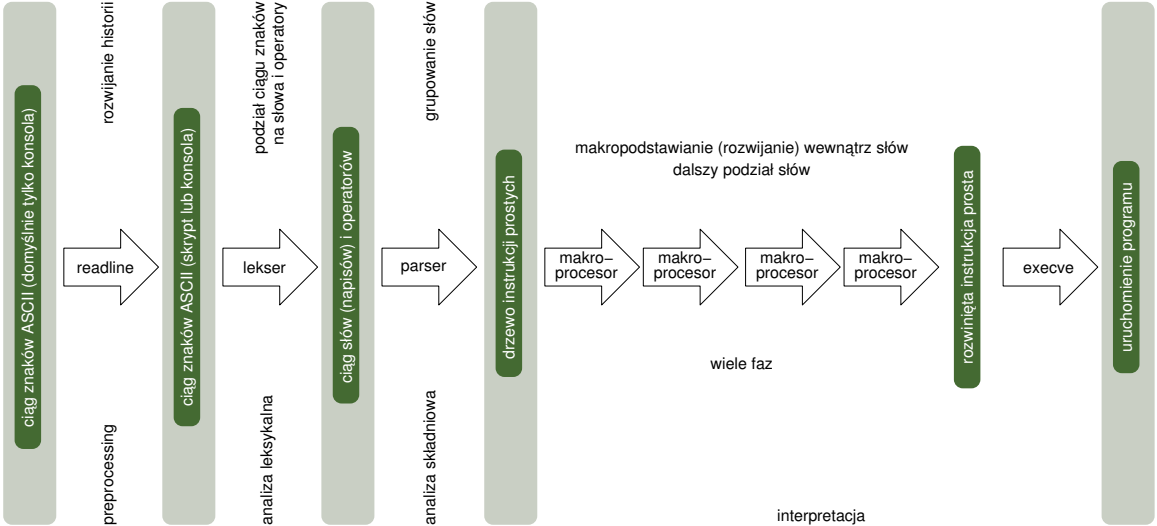
Na tekście instrukcji prostej wybranej do wykonania wykonuje się w kolejności ciąg makropodstawień:

- rozwinięcia nawiasów wąsatych, np. `file{1,2,3}`, `file{1..10}`,
- rozwinięcia tyldy, np. `~/Downloads/`,
- rozwinięcia zmiennych, np. `$HOME`,
- podstawienia instrukcji, np. `$(cat file.txt)`,
- podstawienia procesów, np. `<(pdftops file.pdf -)`,
- rozwinięcia arytmetyczne, np. `$((N+1))`,
- (powtórny) podział słów,
- rozwinięcia nazw plików, np. `file?-*.txt`.

W bashu makrogeneracja odbywa się podczas interpretacji



Tylko `readline` dokonuje preprocesingu



Kolejność rozwijania w bashu

```
$ MYCMD=echo
$ $MYCMD 1 2 3
```

Kolejność rozwijania w bashu

```
$ MYCMD=echo
$ $MYCMD 1 2 3

$ MYKWD=done
$ for i in 1 2 3; do echo $i; $MYKWD
```

Kolejność rozwijania w bashu

```
$ MYCMD=echo
$ $MYCMD 1 2 3

$ MYKWD=done
$ for i in 1 2 3; do echo $i; $MYKWD

$ $MYKWD
bash: done: command not found
$ done
bash: syntax error near unexpected token ‘done’
$ for i in 1 2 3; do echo $i; !!
```

Kolejność rozwijania w bashu

```
$ MYCMD=echo
$ $MYCMD 1 2 3

$ MYKWD=done
$ for i in 1 2 3; do echo $i; $MYKWD

$ $MYKWD
bash: done: command not found
$ done
bash: syntax error near unexpected token ‘done’
$ for i in 1 2 3; do echo $i; !!
```

- Rozwijanie historii odbywa się *przed* kompilacją (to *jest* preprocessing).
- Pozostałe makropodstawienia są wykonywane w słowach wykonywanej instrukcji prostej podczas *interpretacji* skryptu.

Struktura i interpretacja basha

Frontend

- Interaktywny: Biblioteki GNU Readline i GNU History.
- Wsadowy: skrypty.

Kompilacja

- Struktura leksykalna.
- Składnia.

Interpretacja

- Wykonywanie instrukcji złożonych w celu wyboru do wykonania instrukcji prostych.
- Rozwijanie instrukcji prostych.

Backend

- Uruchamianie programów i wykonywanie poleceń wbudowanych.

Polecenia basha w pliku — skrypty

- Domyślnie rozwijanie historii jest wyłączone.
- Bash kompiluje i wykonuje tylko jedną instrukcję na raz.
- Wniosek: w skrypcie mogą być błędy składniowe, które nie zostaną wykryte podczas wykonania!
- Wykonywanie skryptu:

  `$ bash` *`plik-z-programem`*
  `$ bash -c '`*`tekst programu`*`'`
- Można nadać plikowi z programem prawa do wykonania:

  `$ chmod a+x` *`plik-z-programem`*

  i uruchamiać poleceniem

  `$` *`plik-z-programem`*

  jak zwykły program.

#! (*hashbang*, *shebang*)

Uruchomienie skryptu z prawami do wykonania jako programu.

- Pierwszy wiersz skryptu postaci

  #!nazwa-interpretera argumenty

  powoduje wykonanie instrukcji

  nazwa-interpretera argumenty plik-z-programem

- Np. jeśli plik wykonywalny `myprog` zawiera wiersz

  `#!/usr/bin/gawk -f`

  to polecenie

  `$ myprog`

  spowoduje wykonanie programu

  `/usr/bin/gawk -f myprog`

- Konwencja: jeśli *plik-z-programem* nie zawiera *hashbang*, to nie powinien mieć prawa do wykonania, a jego nazwa powinna mieć rozszerzenie .sh. Jeśli zawiera *hashbang*, to powinien mieć prawa do wykonania, a nazwa nie powinna zawierać żadnego rozszerzenia.

Instrukcje złożone — prosty interpretowany język programowania

Niech *instr* oznacza instrukcję, *li* — *listę instrukcji*, *sł* — słowo, *zm* — *zmienną*, *wyr* — wyrażenie arymetyczne, *log* — wyrażenie logiczne, a *wz* — wzorzec.

- Instrukcja grupowania: `{` *li* `}`
- Instrukcja podprocesu: `(` *li* `)`
- Instrukcja koprocesu: `coproc` [*zm*] *instr*
- Instrukcja warunkowa: `if` *li* `; then` *li* [`; elif` *li* `; then` *li*...] [`; else` *li*] `; fi`
- Instrukcja wyboru: `case` *sł* `in` [[(] *wz* [| *wz*] ...) *li* [;;|;&|;;&]] ... `esac`
- Instrukcja zapytania: `select` *zm* [`in` *sł*...]; `do` *li* `; done`
- Instrukcja pętli: [`while` | `until`] *li*; `do` *li* `; done`
- Instrukcja iteracji: `for` *zm* [`in` [*sł* ...] `;`] `do` *li* `; done`
- Instrukcja iteracji arytmetycznej: `for ((` *wyr* `;` *wyr* `;` *wyr* `)) ; do` *li* `; done`
- Wyrażenie arytmetyczne: `((` *wyr* `))`, por. instrukcję `let`
- Wyrażenie logiczne: `[[` *log* `]]`, por. instrukcje `test` i `[`

Za dowolną instrukcją złożoną mogą wystąpić przekierowania (uwaga na jednoznaczność).

Analiza kilku prostych skryptów

- jednolinijkowce
- `monitor`