Kurs administrowania systemem Linux

Zajęcia nr 2: Powłoka systemowa

Instytut Informatyki Uniwersytetu Wrocławskiego

4 marca 2024

Wieloprocesowość w Linuksie

- Jądro uruchamia program `/sbin/init` (PID 1).
- `/sbin/init` uruchamia dalsze procesy, np. `/sbin/getty`, który uruchamia `/bin/login`, który uruchamia `/bin/bash`, który uruchamia dalsze procesy.
- Strona wywołująca:

```
int execve(const char *name, char *const argv[], char *const envp[]);
```

- Strona wywoływana:

```
int main(int argc, char *argv[], char *envp[]);
```

- Strona wywoływana kończy pracę wykonując `return` n w `main`.
  Wartość n (0–255) to *kod powrotu*.
- Strona wywołująca otrzymuje kod powrotu (wykonując `waitpid`).
- Wyrażenia sterujące w instrukcjach warunkowych i pętli w bashu to też instrukcje basha (proste lub złożone).
- Instrukcje warunkowe i pętli podejmują decyzje na podstawie kodu powrotu z wykonania wyrażenia sterującego.
- Uwaga: $n = 0$ — prawda, $n \neq 0$ — fałsz!

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.

Biblioteki GNU Readline i GNU History

- Uniwersalny edytor wiersza poleceń.
- Rozbudowana edycja wiersza ze skrótami klawiszowymi w stylu Emacs-a oraz vi.
- Historia wcześniej wprowadzonych wierszy z rozbudowanymi możliwościami wyszukiwania i edycji.
- Bardzo łatwa integracja z dowolnym programem w C, Perlu, Tcl/Tk, Pythonie i innych językach:

```
input = readline(prompt);
```

- Biblioteki współdzielone: `libreadline.so.7.0` i `libhistory.so.7.0`.
- Pakiety w Debianie: `readline-common`, `readline-doc`, `libreadline7`, `libreadline7-dev`, `libreadline7-dbg`.
- Dokumentacja:
  - `readline(3)`, `history(3)`
  - *The Gnu Readline Library*, Brian Fox and Chet Ramey (Texinfo)
  - *The Gnu History Library*, Brian Fox and Chet Ramey (Texinfo)

Readline

Pliki konfiguracyjne

```
/etc/inputrc
~/.inputrc
```

- Ponad 100 komend edycji wiersza.
- Możliwość dowolnego przypisywania konfiguracji klawiszy do komend.
- Dużo komend, które standardowo nie mają przypisanych kombinacji klawiszy.
- Ponad 40 zmiennych konfiguracyjnych.
- Preprocesor plików konfiguracyjnych: warunkowa kompilacja i dołączanie zawartości plików.

Readline w bashu

- Dodatkowe przypisania kombinacji klawiszy.
- Domyślnie działa w trybie interakcyjnym. Opcja `--noediting`.
- Polecenie `set -o emacs` włącza edycję w stylu emacsa, a `set -o vi` — w stylu vi.

Komendy edycji wiersza

Przesuwanie kursora

C-a, <HOME>	beginning-of-line
C-e, <END>	end-of-line
C-f, <RIGHT>	forward-char
C-b, <LEFT>	backward-char
M-f, C/M-<RIGHT>	forward-word
M-b, C/M-<LEFT>	backward-word
C-l	clear-screen

Historia

<Return>, C-j, C-m	accept-line
C-o	oper-and-get-next
C-x C-e	edit-and-exec
M-C-e	shell-expand-line
M-C-y	yank-nth-arg
M-., M-_	yank-last-arg
C-p, <UP>	previous-history
C-n, <DOWN>	next-history
M-<	beginning-of-history
M->	end-of-history
C-r	reverse-search-history
C-s	forward-search-history
M-p	non-inc-rev-search
M-n	non-inc-fwd-search
M-^	history-expand-line

Edycja

C-d	end-of-file
C-d, <DEL>	delete-char
<BSP>	backward-delete-char
C-v, C-q	quoted-insert
C-t	transpose-chars
M-t	transpose-words
M-u	upcase-word
M-l	downcase-word
M-c	capitalize-word

Komendy edycji wiersza (2)

Usuwanie

C-k	kill-line
C-x <BSP>	backward-kill-line
C-u	unix-line-discard
M-d	kill-word
M-<BSP>	backward-kill-word
C-w	unix-word-rubout
M-\	delete-horizontal-space
C-y	yank
M-y	yank-pop

Automatyczne uzupełnianie

<TAB>	complete
M-?	possible-completions
M-*	insert-completions

Dodatkowe uzupełnianie w bashu

M-/	complete-filename
C-x /	possible-filename-compl
M-~	complete-username
C-x ~	possible-username-compl
M-$	complete-variable
C-x $	possible-variable-compl
M-@	complete-hostname
C-x @	possible-hostname-compl
M-!	complete-command
C-x !	possible-command-compl
M-<TAB>	dynamic-complete-history
M-{	complete-into-braces

Makra

C-x (	start-kbd-macro
C-x)	end-kbd-macro
C-x e	call-last-kbd-macro

Argumenty liczbowe komend

M-0, M-1, ..., M-9, M--	digit-arg

Komendy edycji wiersza (3)

Różne

C-x C-r	re-read-init-file
C-g	abort
M-abc...	do-uppercase-version
<ESC>	prefix-meta
C-_, C-x C-u	undo
M-r	revert-line
M-&	tilde-expand
C-@, M-<SPC>	set-mark
C-x C-x	exch-point-and-mark
C-]	character-search
M-C-]	char-search-backward
M-#	insert-comment
M-g	glob-complete-word
C-x *	glob-expand-word
C-x g	glob-list-expansions
C-x C-v	display-shell-version

Edycja w trybie incremental search

C-s	forward search next
C-r	reverse search next
C-r C-r	recall previous search
C-g	abort search
<ESC>, C-j	terminate search
any other	terminate search and
bash command	execute command
e.g., <RETURN>	

Rozwijanie historii

Składania „wyrażeń historycznych”

- Wybieranie wiersza historii: `!`*n*, `!-`*n*, `!!` (= `!-1`), `!`*str*, `!?`*str*[`?`], `^`*str*`^`*str*`^`, `!#`.
- Wybieranie fragmentu wiersza: `:`*n* ($n \geq 0$), `:`*n*`-`*m*, [`:`]`^` (= `:1`), [`:`]`$`, `:`*n*`*` (= `:`*n*`-$`), `:`*n*`-`, [`:`]*n* (= `:0-`*n*), [`:`]`*`, [`:`]`%`.
- Modyfikatory: `h`, `t`, `r`, `e`, `p`, `q`, `x`, `s/`*old*`/`*new*`/`, `&`, `g`, `G`.

Najczęściej używane odwołania do historii

- `!!` — poprzednie polecenie
- `!$` — ostatnie słowo poprzedniego polecenia (skrót od `!!$`)
- `!:2` — drugi argument poprzedniego polecenia (trzecie słowo)
- `!gcc` — ostatnio wprowadzony wiersz zaczynający się znakami `gcc`

Dobre rady

- Kombinacja klawiszy `M-^` działa jak gpp w C — rozwija historię bez kompilowania wiersza.
- `M-2 M-.` ma ten sam efekt, co wpisanie `!:2` i naciśnięcie `M-^`.
- Znaki `\!` w `PS1` wstawiają numer instrukcji do tekstu zachęty.

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.

#! (*hash-bang*, *she-bang*)

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 *hash-bang*, to nie powinien mieć prawa do wykonania, a jego nazwa powinna mieć rozszerzenie .sh. Jeśli zawiera *hash-bang*, to powinien mieć prawa do wykonania, a nazwa nie powinna zawierać żadnego rozszerzenia.

Struktura leksykalna basha

- Komentarze zaczynają się znakiem `#` i kończą znakiem nowego wiersza (w trybie interaktywnym można wyłączyć).
- Ciąg `\<newline>` jest usuwany.
- *Biały znak*: spacja lub znak tabulacji.
- *Metaznak*: znak nowego wiersza, `|`, `&`, `;`, `(`, `)`, `<`, `>`.
- *Token*: *słowo* (wymaga oddzielenia białymi znakami) lub *operator* (nie wymaga).
- *Słowo*: dowolny ciąg znaków różnych niż białe i metaznaki.
- *Operator*: token zbudowany z jednego lub więcej metaznaków.

Operatory i słowa kluczowe

- Operatory dzielą się na *sterujące* i *przekierowania*.
- *Operator sterujący*: znak nowego wiersza, `||`, `&&`, `&`, `;`, `;;`, `;&`, `;;&`, `|`, `|&`, `(`, `)`.
- *Operator przekierowania*: `<`, `>`, `<<`, `>>`, `<<<`, `<>`, `&>`, `>&`, `<&`, `>&`, `&>>`.
- *Słowo kluczowe*: `!`, `case`, `coproc`, `do`, `done`, `elif`, `else`, `esac`, `fi`, `for`, `function`, `if`, `in`, `select`, `then`, `until`, `while`, `{`, `}`, `time`, `[[`, `]]`. Słowo kluczowe jest zarezerwowane tylko wtedy, gdy nie jest ujęte w cudzysłowy i jest pierwszym słowem instrukcji prostej lub trzecim słowem instrukcji `case` lub `for`.

Quoting

Quoting pozwala tworzyć pojedyncze tokeny zawierające białe znaki i metaznaki:

- *Backslash* `\`: odbiera specjalne znaczenie następnemu znakowi z wyjątkiem `<newline>`.
- Apostrofy `'...'`: odbierają specjalne znaczenie ciągowi znaków.
  Ciąg nie może zawierać `'`.
- Cudzysłowy `"..."`: odbierają specjalne znaczenie ciągowi znaków z wyjątkiem `` ` ``, `$` i `\` (tylko jeśli następuje po nim `` ` ``, `$`, `\`, `"` lub `<newline>`). Zmieniają znaczenie `$*` i `$@`.
- Znaki sterujące w stylu C: `$'...'`. Specjalne znaczenie: `\a`, `\b`, `\e`, `\E`, `\f`, `\n`, `\r`, `\t`, `\v`, `\\`, `\'`, `\"`, `\`*nnn*, `\x`*HH*, `\u`*HHHH*, `\U`*HHHHHHHH*, `\c`*x*.
- Teksty zależne od wersji językowej: `$"..."`.

Składnia basha

- Parser dzieli ciągi słów na zdania zakończone `;` lub `<newline>`.
- Zdania zaczynające się słowem kluczowym są fragmentami instrukcji złożonych.
- Instrukcje proste: [*przypisanie zmiennej* ...] *nazwa-programu* [*argument* ...]
- Potoki: [`time` [`-p`]] [`!`] *instrukcja*$_1$ [[`|`|`&`] *instrukcja*$_2$ ...]
- Listy instrukcji: *potok*$_1$ [[`;`|`&`|`&&`|`||`] *potok*$_2$ ...]
- Instrukcje złożone (zawierają listy instrukcji, zmienne, wzorce, wyrażenia arytmetyczne i wyrażenia logiczne).
- W instrucji prostej lub za instrukcją złożoną mogą wystąpić przekierowania (uwaga na jednoznaczność!).

Wyrażenia, funkcje i zmienne

Wyrażenia

- Wyrażenie arytmetyczne: `(( wyr ))`, por. instrukcję `let`
- Wyrażenie logiczne: `[[ log ]]`, por. instrukcje `test` i `[`

Funkcje

- Składnia: [`function`] *zm* [`()`] *instrukcja złożona*
- Musi wystąpić co najmniej jeden z tokenów `function` i `()`
- Funkcje są rekurencyjne
- Wywołanie funkcji jest instrukcją prostą

Zmienne

- Składnia: ciąg liter, cyfr i znaku `_` nie zaczynający się cyfrą lub zmienna specjalna.
- Zmienne specjalne: `*`, `&`, `#`, `?`, `-`, `$`, `!`, `0`, n ($n > 0$), `_`
- Odwołanie do zmiennej („dereferencja”): `$`[`{`]*zmienna*[`}`]

Rozwinięcia w Bashu

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

1. rozwinięcia nawiasów wąsatych, np. `file{1,2,3}`, `file{1..10}`,
2. 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))`,
3. (powtórny) podział słów,
4. rozwinięcia nazw plików (*globy*), np. `file?-*.txt`.

Rozwinięcia są wykonywane od lewej do prawej.

Zasady rozwijania

Rozwinięcia tylko w jednej instrukcji prostej

- Podstawienia są wykonywane *tylko* w bieżąco *wykonywanej* instrukcji *prostej*:

```
$ cd /usr/share; ((cd ..; echo $PWD)); echo $PWD
/usr
/usr/share
```

- Podobnie, w ciągach instrukcji zagnieżdżonych wewnątrz instrukcji prostej, tj. w konstrukcjach `$(...)` i `<(...)` oraz w wyrażeniach arytmetycznych, tj. w konstrukcji `$((...))`, podstawienia są wykonywane dopiero *podczas wykonywania* każdej z tych instrukcji osobno:

```
$ cd /usr/share; echo "$PWD $(cd ..; echo $PWD)"
/usr/share /usr
$ N=0; echo $((N++, N))
1
```

Zasady rozwijania, cd.

Powtórny podział słów i jego wyłączanie

- Powtórny podział słów jest niezbędny, gdyż inne podstawienia (z wyjątkiem rozwinięć *globów*) wstawiają pojedyncze tokeny:

```
$ DIRS="/usr /var /etc"; ls -d $DIRS; ls -d "$DIRS"
/etc/ /usr/ /var/
/bin/ls: cannot access /usr /var /etc: No such file or directory
```

- Podstawowym zadaniem cudzysłowów "..." jest wyłączenie powtórnego podziału na słowa.
- *Globy* wstawiają wiele tokenów (po jednym dla każdej nazwy pliku, nawet jeśli zawiera spacje) i jako jedyne są rozwijane *po* powtórnym podziale na słowa. Dzięki temu nazwy plików zawierające spacje nie rozpadną się.

```
$ touch 'a b'; ls a*
a b
$ eval ls a*
/bin/ls: cannot access a: No such file or directory
/bin/ls: cannot access b: No such file or directory
```

Rozwinięcia wyliczeniowych nawiasów wąsatych

Wygenerowanie ciągu słów według wzoru:

```
$ echo Klaud{ia,io,yna}
Klaudia Klaudio Klaudyna
```

Polecenie

```
$ mkdir -p /usr/local/share/texmf/{fonts/{{truetype,tfm}/myfont,{map,enc}/pdftex},tex/latex/myfont}
```

utworzy katalogi (co ciekawe, w podanej niżej kolejności):

```
/usr/local/share/texmf/fonts/enc/pdftex/
/usr/local/share/texmf/fonts/map/pdftex/
/usr/local/share/texmf/fonts/tfm/myfont/
/usr/local/share/texmf/fonts/truetype/myfont/
/usr/local/share/texmf/tex/latex/myfont/
```

Rozwinięcia arytmetycznych nawiasów wąsatych

```
$ echo {5..12}
5 6 7 8 9 10 11 12
$ echo {8..1..2}
8 6 4 2
```

Można zagnieżdżać w wyliczeniowych nawiasach wąsatych:

```
$ echo {{5..12},{14..18}}
5 6 7 8 9 10 11 12 14 15 16 17 18
```

Przydatne rozszerzenie (zera wiodące):

```
$ echo {01..15}
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
```

Rozwinięcia nazwiasów wąsatych są wykonywane *przed* główną grupą rozwinięć (podstawieniami zmiennych itp.):

```
$ X1=a; X2=b; X3=c; echo $X{1..3}
a b c
```

Rozwinięcia tyldy

Formy:

- `~` — `$HOME` użytkownika, np. `~/.bashrc` → `/home/user/.bashrc`
- `~`*`otheruser`* — `$HOME` użytkownika *`otheruser`*,
  np. `~jan/.bashrc` → `/home/jan/.bashrc`
- `~+` — `$PWD`
- `~-` — `$OLDPWD`, np.

  ```
  $ cd /tmp; touch a; cd ~; echo $OLDPWD $PWD; ls ~-/a
  /tmp /home/user
  /tmp/a
  ```
- `~`*`N`*, `~+`*`N`*, `~-`*`N`* — zob. polecenie wbudowane `dirs`.