Shell – podstawy syntaksu

Paweł Józiak

Wydział Matematyki i Informatyki Politechniki Warszawskiej

21 X 2024

Plan zajęć

Plan zajęć

Powłoki i komendy

- Większość komend jest zaimplementowanych jako osobne, małe programy,
- Syntaks programu jest niezależny od powłoki i zależy wyłącznie od jego autorów (choć istnieją pewne konwencje),
- Nazwy plików, sprzętów, strumieni etc są również niezależne od powłoki.
- Różne powłoki mogą różnić się:
  - plikami konfiguracyjnymi,
  - dodatkowymi cechami,
  - pewnymi operatorami oraz znakami specjalnymi
  - sposobem definiowania i używania zmiennych
  - komendami wbudowanymi,
  - instrukcjami kontrolującymi.
- wiele różnych powłok może być zainstalowanych na jednym systemie oraz być odpalonymi przez użytkownika w tym samym czasie.

Sufler komend

- Sufler komend (*command prompt*, albo wręcz *prompt*) to ciąg znaków, po którym widnieje mrugający kursor wskazujący, że Powłoka jest gotowa na współpracę z użytkownikiem,
- Przez konwencję, sufler zwykłego użytkownika kończy się symbolem dolara ($) lub procenta (%), zaś sufler użytkownika `root` kończy się krzyżykiem (#).
  ✍ Domyślnie sufler może zawierać nazwę użytkownika, komputera etc; ponadto może być pokolorowany:

Sufler dla użytkowników oraz *roota* w Gentoo Linux

```
me@myhost ~ $
myhost ~ #
```

- W materiałach tych zajęć trzymać będziemy się tej konwencji: symbole $ oraz # będą oznaczać komendy wywoływane przez zwykłego użytkownika, bądź użytkownika *root*, odpowiednio.

Wprowadzanie komend

- Poniższe ogólne zasady znajdują zastosowanie w każdej powłoce Unixa:
  - W nazwach komend, plików, opcje, zmienne itd wielkość liter ma znaczenie,
  - Komendy powinny mieścić się w jednej linii,
  - Pojedynczy *backslash* (\) na końcu linii, bądź niezakończona sekwencja rozpoczęta apostrofem, pozwala na kontynuowanie komendy w kolejnej linii,
  - Krzyżyk (#) sprawia że pozostałe znaki w linii są ignorowane przez interpterer (zakomentowanie),
  - Sekwencje znaków białych uważane są za separator słów,
  - Istnieją sposoby za zachowanie literalnego znaczenia znaków specjalnych.

Bourne shell a C-shell

- *Bourne shell (sh)*:
  - Domyślna powłoka pochodząca od AT&T nim Unix stał się oprogramowaniem komercyjnym,
  - syntaks inspirowany językiem ALGOL 68,
  - bardzo użyteczny w skryptowaniu/programowaniu.
- *C shell (csh)*:
  - Powłoka BSD z pewnymi ulepszeniami (historia, aliasy, uzupełnianie nazw plików etc.),
  - domyślna powłoka w systemach BSD,
  - spójny syntaks wzorowany na języku C,
  - dziś zazwyczaj zastąpiony przez jego rozszerzenie: *tcsh*.

Współczesne powłoki

- Powłoki Bourne'a (zgodne z *sh*):

  Korn shell (ksh) rozszerzenie *sh* o pewne cechy powłoki *csh*,

  Bourne-again shell (bash) składający (*bashing*) wszystkie najlepsze cechy *sh*, *csh* oraz *ksh* (zob. dalej),

  zsh Bogaty w funkcjonalności, ale ciężki do skonfigurowania.

- fish friendly interactive shell; nie będący w pełni zgodny ani z *sh*, ani z *csh*,
- tcsh ulepszona wersja powłoki *csh* (zwykle zastępuje *csh*).

Bourne-again (born-again?) shell

- Podstawowe cechy *basha*:
  - część projektu GNU,
  - domyślna powłoka na niemal wszystkich systemach Linux/GNU, a także MacOS (10.3 - 10.14)
  - dostępny na niemal wszystkich systemach rodziny Unix,
  - realizujący standard POSIX,
  - wstecznie kompatybilny z Bourne shell (*sh*),
  - zawiera wiele użytecznych funkcjonalności powłoki Korna (*ksh*) oraz powłoki C (*csh*).
- Będziemy używać *basha* jako naszej domyślnej powłoki,
- większość informacji z tym kursie będzie działać jednakowo także w pozostałych powłokach.

Plan zajęć

Komendy

- Jako nazwę komendy możemy wpisać jedno z:
  - wewnętrzną funkcję interpretera (tzw. *built-in*),
  - nazwę wykonywalnego pliku *binarnego*,
  - nazwę *skryptu*, wykonywalnego pliku tekstowego który będzie interpretowany przez powłokę
- Z perspektywy użytkownika skrypty, *built-in*y oraz programy binarne są nierozróżnialne.
- Jeśli istnieje program o takiej samej nazwie jak *built-in*, to właśnie *built-in* ma pierwszeństwo,
- Powłoka przeszukuje szereg katalogów w poszukiwaniu wykonywalnych plików (binarnych oraz skryptów). Pierwszy znaleziony jest wówczas wywoływany.

Ogólny syntaks komend

Syntaks komend (najpowszechniejsza konwencja)

```
$ komenda [opcje] [nazwy plików]
```

- Ogólnie przyjęta konwencja mówi o tym, że kolejność opcji jest dowolna, jednak opcje powinny być przed nazwami plików, na których komenda na operować
- Są dwie konwencje na sposób przekazywania opcji:
  - opcje krótkie (syntaks POSIX),
  - opcje długie (syntaks GNU).

Krótkie opcje (POSIX)

- Opcja krótka składa się z myślnika (-) oraz następującej po nim pojedynczej litery.
  ✍ W poniższym przykładzie listujemy (`ls`) zawartość katalogu `/etc` za pomocą rozszerzonego formatu (`-l`):

Wyświetlanie zawartości katalogu /etc w rozszerzonym formacie

```
$ ls -l /etc
```

- Opcje krótkie można zgrupować pod jednym myślnikiem:
  ✍ Wszystkie poniższe komendy są równoważne (opcja `-A` oznacza uwzględnianie plików ukrytych):

Wyświetlanie zawartości katalogu /etc w rozszerzonym formacie

```
$ ls -l -A /etc
$ ls -A -l /etc
$ ls -lA /etc
$ ls -Al /etc
```

Opcje klucz-wartość

- Niektóre opcje wymagają określenia dodatkowych wartości
  ✍ Wyświetlamy uporządkowaną zawartość pliku `/etc/passwd`. Określamy, iż separatorem pól jest dwukropek (`-t :`), porządkujemy względem 3. pola (`-k 3`) używając porządku numerycznego (`-n`):

Sortowanie numeryczne względem 3. pola (rozdzielenie dwukropkiem)

```
$ sort -t : -k 3 -n /etc/passwd
```

- Opcje typu klucz-wartość można grupować pod warunkiem, że wartości dla klucza występują bezpośrednio po nazwaniu klucza, i nie doprowadza to do niejednoznaczności
  ✍ Wszystkie poniższe komendy są równoważne:

```
$ sort -t : -k 3 -n /etc/passwd
$ sort -t: -k3 -n /etc/passwd
$ sort -n -k3 -t: /etc/passwd
$ sort -nk3 -t: /etc/passwd
```

Długie opcje (GNU)

- Opcje długie określane są przez `--nazwę-opcji`.
- Większość komend GNU definiuje równoważniki w formacie krótkim i długim. Można ich wówczas używać zamiennie.
  ✍ Poniższe komendy są równoważne:

Wyświetlanie całej zawartości katalogu

```
$ ls -A /etc
$ ls --almost-all /etc
```

- Opcje krótkie i długie mogą być używane razem, w dowolnej kolejności, jednak opcji długich nie można grupować.
  ✍ W poniższym przykładzie używamy zarówno krótkiej jak i długiej opcji:

Wyświetlanie zawartości katalogu /etc w rozszerzonym formacie

```
$ ls -l --almost-all /etc
```

Długie opcje klucz-wartość

- Długie opcje klucz-wartość definiuje się za pomocą `--nazwa-opcji=wartość` (UWAGA: dookoła symbolu równości nie może być żadnych znaków białych).
- Obowiązkowe wartości dla opcji krótkich są również obowiązkowe dla tożsamych opcji długich.
  ✍ Poniższe komendy są równoważne:

Sortowanie numeryczne względem 3. pola (rozdzielenie dwukropkiem)

```
$ sort -t : -k 3 -n /etc/passwd
$ sort --field-separator=: --key=3 --numeric-sort
/etc/passwd
```

Standardowe opcje GNU

- Zachęca się programistów do zaprojektowania obsługi zarówno długich jak i krótkich opcji.
- Użytkownicy GNU mogą założyć, że poniższe dwie opcje są zdefiniowane dla każdej komendy:
  - `--help`
  - `--version`

Nazwy plików a opcje – niejednoznaczność

- Opcje powinny poprzedzać nazwy plików.
- Zazwyczaj pierwszy parametr komendy, który nie zaczyna się myślnikiem, jest uważany za początek listy nazw plików
- Podwójny myślnik (`--`) może być używany do wyraźnego zaznaczenia końca opcji w razie niejednoznaczności.
  ✍ *touch* tworzy pusty plik o zadanej nazwie, jeśli jeszcze nie istniał.
  Chcielibyśmy stworzyć plik o nazwie `--help`:

Podwójny myślnik jako znacznik końca opcji

```
$ touch --help
```
(wyświetla informacje o komendzie touch)
```
$ touch -- --help # '--help' jest teraz nazwą pliku
```

Nazwy plików

- Jeśli to ma sens dla komendy, nazwę pliku można zastąpić listą plików bądź katalogów.
- Jeśli żaden plik nie zostanie nazwany, niektóre komendy zakładają, że chodzi o bieżący katalog.
- Wiele komend, w razie braku podania nazwy pliku, operuje na standardowym wejściu (domyślnie: czyta z klawiatury) oraz standardowym wyjściu (domyślnie: ekranie) dla zapisu.

Plan zajęć

Historia

- Bash zachowuje historię komend wpisanych przez użytkownika. Za pomocą klawiszy ↑ oraz ↓ można szybko przez nie nawigować.
- Szybkie przeszukiwanie historii można także wykonać za pomocą kombinacji ⟨CTRL⟩ + R (i wprowadzeniu ciągu znaków). Odnalezienie wcześniejszych trafień wymaga ponownego naciśnięcia ⟨CTRL⟩ + R.
- Wywołanie *history* pozwoli na wyświetlenie ponumerowanej historii ostatnich komend. Wskazanie jej numeru pozwoli odpalić jedną z nich: !*numer*. Skrót !! reprezentuje ostatnią komendę.
  ✍ Ostatnie dwie komendy są równoważne

Wywołanie historycznej komendy numerem

```
$ history
$ 1 ls -lA /etc
$ 2 sort -t : -k 3 -n /etc/passwd
$ !2
$ sort -t : -k 3 -n /etc/passwd
```

Czyszczenie historii

- Historia bieżącej sesji basha jest cache'owana w pamięci podręcznej.
- Usunięcie historii bieżącej sesji jest osiągalne komendą `history -c`.
- Przy zamknięciu sesji, historia sesji jest dopisywana do pliku z całościową historią (`.bash_history`) w katalogu domowym użytkownika.
- Po usunięciu historii z pamięci podręcznej, możemy chcieć również usunąć zawartość pliku z historią komend.
  ✍ Nie usuwamy pliku. Jedynie wymazujemy jego zawartość:

Czyszczenie historii basha

```
$ echo -n > ~/.bash_history
```

Autouzupełnianie

- Można rozpocząć wpisywanie słowa i nacisnąć przycisk ⟨TAB⟩. Bash postara się uzupełnić słowo zgodnie z następującymi regułami:
  - pierwsze słowo jest uzupełniane do komendy (o ile takie uzupełnienie jest możliwe),
  - kolejne słowa są uzupełniane do możliwych ścieżek plików,
  - w razie niejednoznaczności, dwukrotne naciśnięcie ⟨TAB⟩ pozwoli na wyświetlenie wszystkich możliwych uzupełnień.
    ✍ Poniższe pary komend dają jednakowe wyniki:

Używanie autouzupełniania

```
$ his⟨TAB⟩
$ history
$ ls -l /et⟨TAB⟩
$ ls -l /etc
```

Plan zajęć

Złożone komendy

- Komendy rozdzielone średnikiem (`;`) w jednej linii będą odpalone sekwencyjnie.
- Każda komenda zwraca pewną wartość reprezentującą sukces bądź porażkę wywołania. Komendy można wykonywać sekwencyjnie warunkowo – pod warunkiem zwrócenia sukcesu (`&&`) bądź porażki (`||`) poprzedniej komendy.
  ✍ `cp` kopiuje plik do pliku o zadanej nowej nazwie, `mv` – zmienia (przesuwa) nazwę pliku, `rm` – usuwa plik. Usuwamy `file3` albo kopiujemy `file5` w zależności od wyniku kopiowania:

Sekwencje komend

```
$ cp file1 file2 ; rm file1
$ cp file3 file4 && rm file3
$ cp file5 file6 || mv file5 file6
```

Status końcowy

- Każda komenda (a dokładniej: proces) zwraca pewien status końcowy po zakończeniu wywołania.
- Odwrotnie do intuicji z matematyki/programowania, 0 reprezentuje sukces (logiczną prawdę). Sukces nie wymaga dalszych wyjaśnień.
- Pozostałe wartości reprezentują porażkę (logiczny fałsz). Wartość statusu końcowego może reprezentować rodzaj błędu związanego z danym wywołaniem komendy

Jokery, ścieżki relatywne

- Poniższe symbole mogą być używane przy określaniu ścieżek oraz nazw plików

kropka (.)	bieżący folder,
podwójna kropka (..)	folder nadrzędny,
tylda (~)	folder domowy,
pytajnik (?)	dowolny znak,
gwiazdka (*)	dowolną sekwencję znaków (tekst).

✍ Listowanie wszystkich plików o nazwach dwuliterowych w bieżącym katalogu:

Używanie jokerów

```
$ ls ??
```

Ścieżki absolutne, ścieżki względne

- Ścieżki zaczynające się od / oznaczają ścieżki absolutne (relatywnie jedynie względem korzenia drzewa katalogów).
  ✍ Ta linia będzie działać jednakowo niezależnie od miejsca, z którego zostanie wykonana

Ścieżka absolutna

```
$ ls /somepath
```

- Wszystkie pozostałe ścieżki są ścieżkami relatywnym względem bieżącego katalogu
  ✍ Poniższe komendy są równoważne:

Ścieżki względne

```
$ ls somepath
$ ls ./somepath
```

Dosłowne znacznie pojedynczych znaków

- Każdy pojedynczy znak poprzedzony *backslash*em (\) jest interpretowany przez powłokę dosłownie – *backslash* usuwa znaczenie specjalne. Tę procedurę często nazywa się *escaping*.
  ✍ Jest możliwym (choć wysoce niestosownym) nazwanie pliku * albo nawet '. Opakowanie nazwy takiego pliku w apostrofy albo usunięcie specjalnego znaczenia znaku jest konieczne by móc wskazać na taki plik. W ten sposób również chronimy białe znaki w nazwach plików przed interpretowaniem jako separator kolejnych nazw plików. W poniższym przykładzie # rozpoczyna komentarz.

Usuwanie znaczeń specjalnych znaków

```
$ rm * # usuwa wszystkie pliki
$ rm \* # usuwa pojedynczy plik '*'
$ rm plik ze spacjami # usuwa trzy pliki
$ rm plik\ ze\ spacjami # usuwa pojedynczy plik
```

Pojedyncze apostrofy

- Pojedyncze apostrofy (') pozwalają zachować znaczenie dosłowne wszystkich znaków, które obejmują.
  ✍ Pojedyncze apostrofy pozwalają na jasne wyróżnienie, jeśli istnieje wiele znaków specjalnych, których znaczenie chcemy usunąć.

Używanie pojedynczych apostrofów

```
$ rm a\ b\ c\ d\ e # usuwa pojedynczy plik
$ rm 'a b c d e' # usuwa pojedynczy plik
```

Podwójne apostrofy

- Podwójny apostrof (") zachowuje znaczenie dosłowne wszystkich znaków, które obejmują, z wyłączeniem \, $ oraz ‘ (na klawiszu z tyldą):
  - $ pozwala na używanie zmiennych,
  - ‘ pozwala na tzw. natychmiastowe wywołanie komendy (*in-line command execution* albo *command substitution*) – zob. prezentacja 4.
- Ze względu na specifyczny syntaks, niektóre inne znaki specjalne mogą zachować specjalne znaczenie, jeśli używane w połączeniu z $.

Znaki specjalne w nazwach plików

- Listując zawartość katalogu, nazwy plików ze znakami specjalnymi mogą być wyświetlane za pomocą pojedynczych apostrofów.
- Nie ma możliwości, by wyświetlić pojedynczy apostrof wewnętrz pojedynczego apostrofu.
- Listując zawartość katalogu, nazwy plików z pojedynczym apostrofem mogą być wyświetlane za pomocą podwójnych apostrofów.
  ✍ Używanie podwójnych apostrofów.

Jak nazywają się pliki?

```
$ touch \"
  ’"’
$ touch \’
  "’"
```

Uwagi o usuwaniu specjalnych znaczeń i apostrofach

- Usuwanie specjalnych znaczeń za pomocą \oraz apostrofów wpływa tylko na sposób czytania znaków przez powłokę
- Jest to transperentne dla komend
  ✍ We wszystkich przypadkach `touch` widzi jedynie `--help`.

Usuwanie specjalnych znaków jest dla powłoki, nie dla komend

```
$ touch --help
$ touch "--help"
$ touch '--help'
$ touch \-\-help
```

- Apostrofy są parsowane od lewej do prawej, bez zagnieżdżeń.
  ✍ Pojedyncze apostrofy nic tu nie znaczą.

Parsowanie apostrofów od lewej do prawej

```
$ echo "'\$'"
  '$'
```

Komendy w wielu liniach

- Kończenie linii za pomocą \usuwamy specjalne znacznie znaku końca linii.
- Parowanie apostrofów jest konieczne – pozostawienie niesparowanych apostrofów pozwala na rozciągnięcie komendy na wiele linii.

Plan zajęć

Zmienne jako etykiety

- W wielu (wszystkich?) językach skryptowych zmienne używane są jako etykiety obiektów.
- Zmienne same w sobie są nietypowane, i nie muszą być deklarowane (choć w niektórych językach byty, które stoją za danymi etykietami, mogą być typowane). W powłoce istnieje jedynie jeden typ zmiennej – tekstowy (*string*).
- Wywołanie niezdefiniowanej zmiennej nie skutkuje błędem w Powłoce – zamiast tego wartość domyślna, pusty teskt, jest zwracana.

Po co zmienne?

- Powłoka to interpreter swojego języka programowania.
- Zmienne mogą przechowywać długie literały i uprościć dzięki temu notacje dla komend.
- Zmiennych użyć można do dostosowania Powłoki.
- Tak zwane zmienne środowiskowe mogą być dziedziczone przez procesy pochodne (ang. *child processes*).

Zmienne w Bourne Shell

- Niniejsze informacje dotyczą do Powłok zgodnych z *Bourne shell*. Powłoki pochodzące od *C-shell* używają nieco innego syntaksu, zgodnego z językiem programowania C.
- Choć nie jest to obowiązek, praktykuje się nazywanie zmiennych dużymi literami.
- Zmienne deklaruje się przez przypisanie im wartości: *VARIABLE_NAME*=*value* (UWAGA: dookoła znaku równości "=" nie może być żadnych białych znaków)
- Aby wywołać wartość zmiennej, używamy symbolu dolara po którym podajemy nazwę zmiennej: $*VARIABLE_NAME*. Dla jednoznaczności, można zamknąć nazwę zmiennej w nawiasach wąsatych: ${*VARIABLE_NAME*} (por. następny slajd).

Zarządzanie zmiennymi

Ustawienie wartości zmiennej

$ SOMEVAR=“something”

Wywołanie wartości zmiennej

$ echo $SOMEVAR
$ echo ${SOMEVAR}

Wyświetlanie wszystkich zmiennych

$ set

Usunięcie zmiennej

$ unset SOMEVAR
$ SOMEVAR=

Zmienne – przykłady

- *echo* jest odpowiednikiem komendy *print*.

Zarządzanie zmiennymi

```
$ VAR1=/etc
$ echo $VAR1
  /etc
$ ls -lA $VAR1
 (wyświetla zawartość katalogu /etc)
$ unset VAR1
$ echo $VAR1
```

- Nie spodziewamy się zmiennej *VAR2ANGENS*

Wąsate nawiasy by uniknąć niejasności

```
$ VAR2=KOT
$ echo ${VAR2}ANGENS
  KOTANGENS
```

Historia Basha – powtórka

- Historia Basha kontrolowana jest za pomocą następujących zmiennych:

HISTSIZE	liczba ostatnich komend zapamiętywanych w cache'u historii,
HISTFILE	ścieżka pliku, w którym przechowywana jest historia,
HISTFILESIZE	liczba komend zapisywanych do pliku historii.

Środowisko

- Dla każdego procesu zestaw zmiennych określa jego środowisko.
- Procesy pochodne (procesy uruchamiane przez bieżący proces) otrzymują kopię środowiska procesu tworzącego podczas konstrukcji.
- Komenda *export* pozwala na dodanie zmiennej powłoki do środowiska procesu.

✍ Można wyeksportować istniejącą zmienną bądź połączyć nadawanie wartości i eksport zmiennej.

Eksport zmiennych

```
$ VARIABLE1=value1
$ export VARIABLE1
$ export VARIABLE2=value2
```

- *printenv* oraz *env* drukuje na ekran listę zmiennych środowiskowych,
- Bash zawiera dodatkowe funkcje wbudowane, które pozwalają na przeglądanie i zarządzanie atrybutami (statusami) wszystkich zmiennych.

Przegląd ważnych zmiennych

- Niektóre najważniejsze zmienne systemowe to:

HOME	ścieżka do katalogu domowego użytkownika,
LC_*, LANG	ustawienia lokalizacyjne,
PATH	rozdzielona dwukropkiem lista katalogów, w których następuje szukanie plików wykonywalnych,
PWD	ścieżka do obecnego katalogu,
SHELL	ścieżka do pliku wykonywalnego powłoki.

- Inne zmienne powłoki to m.in.

IFS	separator słów (domyślnie: białe znaki),
PS1	*primary prompt description* – format suflera.

Zmiana zmiennych środowiskowych – przykłady

✍ Zmiana domyślnego formatu suflera:

```
$ PS1='Hello>'
Hello>
```

✍ Dodanie bieżącej ścieżki jako pierwszej w liście katalogów przeszukiwalnych pod kątem plików wykonywalnych:

```
$ PATH=.:$PATH
```

✍ Wybranie języka interfejsu (należy go dokonać przez wystartowaniem GUI):

```
$ export LANG=pl_PL.UTF-8
```

Plan zajęć

Systemowe pliki konfiguracyjne vs pliki konfiguracyjne użytkownika

- Większość globalnych plików konfiguracyjnych systemu jest umiejscowionych w katalogu */etc* (*edit-to-configure*).
- Większość plików konfiguracyjnych używa syntaksu *przyjaznego użytkownikom Basha*:
  - są to pliki tekstowe ASCII,
  - każda linia uważana jest za osobny wpis,
  - # reprezentuje komentarz,
  - nastawienie wartości zmiennych odbywa się jak w Powłoce.
- Większość ustawień ogólnosystemowych może być nadpisanych przez ustawienia użytkownika. Pliki konfiguracyjne użytkownika leżą w katalogu domowym użytkownika, a ich nazwy zaczynają się od . (dzięki czemu są traktowane jako pliki ukryte).

Pliki konfiguracyjne Basha

- Pliki inicjalizacyjne powłoki używają podobych konwencji nazewniczych. Nie ma jednak reguły – nawet pliki konfiguracyjne dla Basha mogą mieć nieco inne nazwy w różnych dystrybucjach Linuxa/BSD. Dla ustalenia właściwych reguł, zerknij do dokumentacji Powłoki w danym systemie.
- Tak wygląda lista plików konfiguracyjnych z dokumentacji Basha:

/etc/profile	Ogólnosystemowy plik instancjalizacyjny powłoki, odpalany w każdej powłoce po zalogowaniu.
~/.bash_profile	Personalny plik instancjalizacyjny, odpalany w każdej powłoce po zalogowaniu danego użytkownika.
~/.bashrc	Personalny plik instancjalizacyjny, odpalany *per-interactive-shell*.
~/.bash_logout	Personalny plik z sekwencją instrukcji odpalanych przy zamykaniu powłoki.