SOWID – PODSTAWY SYNTAKSU POWŁOKI

PAWEŁ JÓZIAK
WYDZIAŁ MATEMATYKI I NAUK INFORMACYJNYCH
POLITECHNIKI WARSZAWSKIEJ

- Nigdy nie wykonuj poleceń metodą *copy&paste*. Przepisz je ręcznie!
- Nie redukuj pracy nad zadaniem do przeczytania go. Wykonaj je i upewnij się, że rozumiesz jak i dlaczego wynik jest taki, jak na ekranie. Zajrzyj do prezentacji bądź zapytaj prowadzącego, jeśli coś nie będzie jasne.
- Linia zaczynająca się od $ reprezentuje prompt zwykłego użytkownika (nie przepisuj go!). Komendą jest to, co następuje po nim.
- Ostrożnie ze spacjami: nie dodawaj ich więcej ani nie usuwaj takowych, jeśli są!

1. Powłoki

(a) Jaka wersja `bash`a jest zainstalowana na Twojej stacji roboczej? Sprawdźmy:
```
$ bash --version
```
(b) Domyślną powłoką na naszych stacjach roboczych jest `bash`. Czy są inne nowoczesne powłoki zainstalowane na Twojej stacji roboczej: `tcsh, ksh, zsh`? Spróbuj je odpalić, na przykład:
```
$ tcsh
```
Aby wyjść, wywołaj komendę:
```
$ exit
```
Zwróć uwagę, że domyślne prompty mogą wyglądać inaczej w różnych powłokach.

(c) Możesz zredefiniować swój domyślny prompt w jakiejkolwiek powłoce zgodnej z Bourne-shell wywołując:
```
$ PS1="To mój nowy sufler "
```
Pamiętaj o tym, by dookoła "=" nie było żadnych białych znaków!
Zamknij terminal i otwórz nowy, dzięki czemu format prompta powróci do domyślnej (znacznie bardziej użytecznej) postaci.

2. Wprowadzanie komend

(a) Sprawdź, że dla komend i opcji wielkość liter ma znaczenie:
```
$ ls
$ ls --version
```
Te nie zadziałają:
```
$ Ls
$ ls --Version
```
(b) Pamiętaj, że oddzielenie białymi znakami jest konieczne. Ta komenda nie zadziała:
```
$ ls--version
```
(c) Zwróć uwagę, że # w zmiennej tekstowej nie startuje komentarza. Sprawdź poniższe:
```
$ echo "Hello"
$ #echo "Hello"
$ echo #"Hello"
$ echo "#Hello"
```
(d) Niezakończone zmienne tekstowe są niedozwolone. Użytkownik będzie proszony o kontynuowanie w nowej linii:
```
$ echo \
>"Hello"
```

```
$ echo "Hello
>"
```
W razie problemów możesz spróbować ponowić uruchomienie emulatora terminala. Przydatna może być również kombinacja `CTRL+C`.

3. Uruchamianie prostych komend

(a) Komenda `'echo'` jest dwukrotnie zaimplementowana: jako *built-in* `bash`a oraz jako program binarny:
```
$ echo "Czołem!"
$ /bin/echo "Czołem!"
```
Działają w zasadzie identycznie, jednak w bardzo rzadkich przypadkach mogą istnieć różnice. Sprawdź
```
$ echo --help
$ /bin/echo --help
```
Szczerze mówiąc, taka implementacja jest podatna na błędy i może doprowadzić do nieprzewidzianych efektów ubocznych. Innymi słowy: błąd projektowy.

(b) Odpal komendy ze slajdów 13-16.

(c) Odpal pomoc komendy `ls`:
```
$ ls --help
```
Jaka jest długa opcja równoważna `'-l'`?
Zwróć uwagę na informację o statusie końcowym (*exit status*) – na końcu. Pamiętaj, 0 oznacza sukces/prawdę!

(d) Wyświetl pomoc komendy `touch`:
```
$ touch --help
```

(e) Utwórz plik o nazwie `--help`:
```
$ touch -- --help
```
oraz usuń go:
```
rm -- --help
```

(f) Utwórz trzy pliki `'plik1'`, `'plik2'`, `'plik3'` za pomocą jednej komendy:
```
$ touch plik1 plik2 plik3
```
oraz usuń je za jednym zamachem:
```
$ rm plik1 plik2 plik3
```

4. Własności basha

(a) Wyświetl wszystkie komendy rozpoznawane przez Twoją powłokę, zaczynające się na `'b'`. Jak z literą `'c'`?
Wskazówka: Użyj do tego autouzupełniania. Wciśnij <TAB >dwukrotnie, jeśli to konieczne. Naciśnij <SPACE >jeśli zobaczysz `'--More--'`.

(b) Jak dużo plików w `'/etc'` zaczyna się na `'pa'`? Pamiętaj: jedynie argumenty komend są uzupełniane do nazw plików, więc wpisz uprzednio nazwę jakiejś (jakiejkolwiek) komendy używanej na zajęciach.

(c) Eksperyment z *historią* `bash`a: wywołaj poniższą komendę
```
$ history | grep pewienwzorzec
```
by zobaczyć jedynie te linie historii, które pasują do wzorca wyszukiwania *pewienwzorzec*. Warto zauważyć, że ten modyfikator komend (tzw. filtr) działa także dla innych komend.

(d) Co robi komenda `!!` – sprawdź!

(e) Nawiguj w `bash`u do swojego katalogu domowego, jeśli tam już nie jesteś:
```
$ cd
$ cat .bash_history
```
Wyczyść historię bieżącej sesji `bash`a:
```
$ history -c
```
Usuń zawartość Twojego pliku z historią (`.bash_history`) – opróżnij ją permanentnie:
```
$ echo -n > .bash_history
```

5. Znaki specjalne

(a) Zmień katalog i wyświetl ścieżkę nowego katalogu:
```
$ cd /etc; pwd
$ cd /niematakiego; pwd
```
Nie wyświetlaj ścieżki do nowego katalogu jeśli zmiena katalogu się nie udała:
```
$ cd /niematakiego && pwd
```
Jeśli nie możesz wejść do nieistniejącego katalogu, wówczas wejdź do katalogu domowego (`cd` bez argumentów przechodzi do katalogu domowego):
```
$ cd /niematakiego || cd && pwd
```
Pamiętaj, że komendy są ewaluowane od lewej do prawej.

(b) Zawartość jakiego katalogu będzie wyświetlona w wyniku wywołania poniższej komendy:
```
$ ls /./etc/./../usr/../././
```
Cóż, pojedyncza kropka nic nie zmienia, więc możemy skrócić to do:
```
$ ls /etc/../usr/../
```
Dokonujemy wejścia do podfolderu głównego folderu i przejście do katalogu wyżej , więc jest to równoważne z:
```
$ ls /
```

(c) Czy te komendy są równoważne?
```
$ ls .
$ ls
```
Owszem, są.

(d) Używając np. *Geany* napisz prosty program typu *hello world* i skompiluj go (ikonka cegły). Otwórz emulator terminalu. Czy te dwie komendy są równoważne?
```
$ ./myprogram
$ myprogram
```
Nie, nie są. Jeśli ścieżka nie jest podana absolutnie, powłoka poszukuje plików binarnych jedynie w katalogach wymienionych w liście `PATH`. Twój katalog domowy nie jest na tej liście. W pierwszym przypadku podana została ścieżka (relatywna). W tym drugim przypadku – żadna ścieżka nie została podana.

(e) Czy te komendy są równoważne?
```
$ ls /etc
$ ls etc
```
Nie, nie są. Ta pierwsza specyfikuje ścieżkę absolutną. Ta druga próbuje wylistować zawartość podkatalogu `'etc'` katalogu, z którego odpalana jest komenda

(f) Utwórz kilka plików o podobnych nazwach, na przykład:
```
$ touch f1 f2 f3 f4 f5 f6
```
i usuń je za pomocą jokera:
```
$ rm f?
```

(g) Utwórz nowy katalog (`mkdir`) i wejdź doń na potrzeby tego ćwiczenia:
```
$ mkdir SOMEDIR; cd SOMEDIR
```
Spróbuj zgadnąć i sprawdź jakie pliki będą utworzone za pomocą poniższych komend:
```
$ touch \$ ; ls
$ touch "'" ; ls
$ touch \~ ; ls
$ touch '\*' ; ls
$ touch "\"\"" ; ls
$ touch \# ; ls
```
Kontynuuj eksperyment samodzielnie.
W końcu wejdź do swojego katalogu domowego i usuń `SOMEDIR` wraz z zawartością:
```
$ cd .. ; rm -r SOMEDIR
```

(h) Utwórz plik o nazwie zawierającej spacje:
```
$ touch "file name with spaces"
```

Zacznij wpisywać a potem naciśnij <TAB>dla autouzupełniania. Co zostanie uzupełnione?

```
$ ls file
$ ls "file
```

6. Zmienne

(a) Pamiętaj że wokół znaku równości nie może być separatorów:
```
$ SOMEVAR="something"
```
Co się stanie jesli jednak wstawisz tam spacje?
```
$ SOMEVAR = "something"
```

(b) Czy odwołanie do niezadeklarowanej zmiennej skutkuje błędem?
```
$ echo $UNASSIGNED
```
Absolutnie nie. Te są traktowane jako zmienne przechowujące pusty tekst.

(c) Czy zmienne mogą przechowywać dowolny tekst, i być interpretowane? Owszem:
```
$ SOMEVAR=ls
$ $SOMEVAR
$ SOMEVAR=l
$ ${SOMEVAR}s
```

(d) Sprawdź, czy `'~'` oraz `'HOME'` wskazują na jednakową lokalizację:
```
$ echo $HOME
$ echo ~
```
Owszem, tak właśnie jest.

(e) Sprawdź, czy `'~'` oraz `'HOME'` w podwójnych cudzysłowach wskazują na jednakową lokalizację:
```
$ echo "$HOME"
$ echo "~"
```
Nie jest tak. Wynika to z faktu, że o ile `$` jest interpretowany wewnątrz podwójnego apostrofu (”), o tyle tylda (˜) nie jest.

(f) Jak dużo spośród zmiennych powłoki są jednocześnie zmiennymi środowiskowymi (`'| sort'` sortuje linie alfabetycznie)? Sprawdźmy:
```
$ set
$ env | sort
```

(g) Sprawdź wartości ważnych zmiennych ze slajdu nr 43.

(h) Czy nowe zmienne są dziedziczone przez procesu utworzony?
```
$ SOMENEW="something"
$ bash
$ echo $SOMENEW
```
Nie, nie są, jeśli ich nie wyeksportujemy.

(i) Czy zmienna `LANG` jest dziedziczona przez proces utworzony?
```
$ LANG=pl_PL.UTF-8
$ bash --version
```
Owszem, jest, gdyż jest to zmienna środowiskowa.

(j) Dodaj bieżący folder (czyli `'.'`) do zmiennej `PATH`:
```
$ PATH=${PATH}:.
```
albo, alternatywnie:
```
$ PATH=.:${PATH}
```
I sprawdź to:
```
$ echo $PATH
```

(k) Przy założeniach punktu (6j), wykonaj ponownie ćwiczenie (5d).

7. Konfigurowanie historii basha

(a) Niech bash pamięta więcej komend z historii. Otwórz plik `'.bashrc'` w swoim katalogu domowym za pomocą edytora tekstu lub utwórz, jeśli nie istnieje. Dodaj poniższe linie na końcu (bądź znajdź podobne linie w istniejącym pliku):
```
HISTSIZE=2048
HISTFILESIZE=2048
```
Będzie działać dla wszystkich instancji powłoki bash wystartowanych począwszy od teraz.