Systemy Operacyjne, notka c3:

Zakres materiału w tym dokumencie:
<> zmienne
<> skrypty
<> przekierowania

1. **Wymień a następnie krótko zdefiniuj rodzaje zmiennych w powłoce.**
   Odp. Wyróżniamy:
   <> zmienne programowe(użytkownika)
   Są to zmienne definiowane przez użytkownika w skryptach powłoki. Definicja zmiennej programowej wygląda następująco: nazwazmiennej=wartość (uwaga: bez spacji po i przed znakiem =)!!! np. imie=czarek(imie to zmienna a czarek to wartość jaką przypisujemy do zmiennej imie, pamiętasz nie???). Zmienne programowe zapisujemy małymi literami! Należy też pamiętać, że są to tzw. zmienne lokalne czyli po wyłączeniu powłoki zostaną one usunięte.

   <> zmienne środowiskowe(globalne)
   Są to zmienne używane do przekazywania informacji powłoce i progamom uruchamianym pod jej kontrolą. Informacje zapisane w zmiennych środowiskowych są wykorzystywane do określenia parametrów interakcji pomiędzy użytkownikiem a powłoką, systemem operacyjnym i programami. Zmienne środowiskowe zapisujemy wielkimi literami! Np. ILE=informacje. Wiele zmiennych środowiskowych definiowanych jest w globalnych plikach konfiguracyjnych. Takie zmienne są pamiętane(DO SPRAWDZENIA W POWŁOCE!)

   <> zmienne specjalne(systemowe)
   Są to zmienne zdefiniowane przez system. Nie możemy zmieniać ich wartości, możemy tylko je przeczytać! Najczęściej są wykorzystywane w skryptach do sterowania programem. Przykładowe zmienne specjalne to $# , $?

2. **Polecenie echo – co robi?**
   Odp. Wypisuje na terminalu tekst, działanie tego polecenia jest podobne do tego cout w języku C++

3. **Czy powłoka wymaga deklarowania typu zmiennej? Prawda/Fałsz**
   Odp. Fałsz, powłowa nie wymaga deklarowania typu zmiennej – no nie trzeba, co tu dużo gadać.

4. **W powłoce Bash wszystkie zmienne są traktowane jako napisy.**
   Prawda/Fałsz
   Odp. Prawda.

5. **x=123, jakiego rodzaju to zmienna?**
   Odp. Jest to zmienna użytkownika,programowa. Zwróć uwagę, że nie ma odstępów przy znaku =! Przypisujemy do zmiennej x napis 123!

6. **Jakie polecenie pozwala usunąć zmienną?**
   Odp. Polecenie unset nazwa_zmiennej pozwala usunąć zmienną.

7. **Jaki znak pozwala odwołać się do wartości przechowywanej przez zmienną? Czy musi on zawsze stać przy nawiasie klamrowym?**
   Odp. $ pozwala odwołać się do wartości przechowywanej przez zmienną i nie musi zawsze stać przy nawiasie klamrowym, może też być bez ale wtedy trzeba uważać!

8. **W jaki sposób wyświetlić wszystkie zmienne powłoki?**
   Odp. Użyję polecenia set bądź env, wyświetli ono wszystkie zmienne powłoki.

9. **Dokończ zdanie: Małymi literami zapisujemy zmienne …….**
   Odp. programowe(użytkownika). Zmienne programowe/użytkownika zapisujemy małymi literami, np. ile, cud, omega – to są zmienne programowe/użytkownika.

10. **Wiesz, że x=5, czy polecenie x=x+7 spowoduje, że za x będzie 12?**
    Prawda/Fałsz
    Odp. Fałsz, wiemy, że wszystkie zmienne są trakowane jako napisy więc jeśli napiszemy x=x+7 to powłoka widzi x+7 jako napis a nie wyrażenie arytmetyczne. Aby zinterpretować to jako wyrażenie arytmetyczne trzeba użyć jednego z dwóch sposobów.

11. **Jak możemy zinterpretować wyrażenie arytmetyczne w powłoce(podaj 2 możliwości i przykłady do nich)**
    Odp. Możemy wykorzystać polecenie let aby dokonać takiej interpretacji, np. mając x=5 możemy zrobić taki let „x=x+5”, ten cudzysłów jest tutaj istotny! Możemy też wykorzystać znak $ i nawias, to mając np. x=5, x+5 zapiszemy tak: x=$((x+5))

12. **W powłoce dzielenie jest wykonywane na liczbach całkowitych.**
    Prawda/Fałsz
    Odp.Prawda. Przykład: Gdy 4/3 to powłoka zwróci 1. A tak na marginesie warto pamiętać, że mamy w powłoce oprację modulo symbol: % i potęgowanie symbol: ** no i oczywiście +, -, *.

13. **Co spowoduje wykonanie polecenia: *readonly x=5*. Czy będziemy mogli za tę zmienną podstawić inną wartość? Czy możemy ją usunąć?**
    Odp.Ustawimy zmienną użytkownika/programową jako tylko do odczytu. Nie będziemy mogli za tą zmienną podstawić innej wartości ani nie będziemy mogli jej usunąć. Dopiero po wyłączeniu powłoki ta zmienna zostanie usunięta.

14. **Jakie polecenie służy do pobierania danych od użytkownika i wstawiania ich do zmiennej zdefiniowanej przez użytkownika?**
    Odp. Polecenie read służy do pobierania danych użytkownika i wstawiania ich do zmiennej, często używamy z tym poleceniem argumentu –p żeby dodać komentarz np. read –p „Podaj ile masz lat" wiek, gdzie wiek to zmienna która przechowa wartość podaną przez użytkownika!

15. **Zmienne globalne to zmienne środowiskowe.**
    Prawda/Fałsz
    Odp. Prawda. Zmienne środowiskowe inaczej nazywamy zmiennymi globalnymi.

16. **Dokończ zdanie: Dużymi literami zapisujemy zmienne …….**
    Odp. zmienne środowiskowe(globalne). To jest taka przyjęta umowa w powłoce o wielkich literach dla zmiennej środowiskowej i małych literach dla zmiennej programowej(użytkownika), nic się nie stanie jak napiszemy małymi literami ale tak się ustaliło i należy tego przestrzegać, żeby nie mieć potem kłopotów z interpretacją czy jest to zmienna środowiskowa czy użytkowa. Potem jak jakaś osoba zerka nasz kod, to wie, że zmienna ile jest zmienną programową(użytkownika). a zmienna CUD jest zmienną środowiskową(globalną).

17. **Zdefiniuj te zmienne(oczywiście są to zmienne środowiskowe, pamiętasz)**
    a) **HISTFILE –** plik historii poleceń, możemy dzięki niemu używać strzałki aby odtworzyć polecenia.
    b) **HISTSIZE –** rozmiar pliku liczony w wierszach, który przechowuje wypisane przez nas polecenia, jeśli przekroczy maksymalną ilość wierszy (1000) to jest kasowany i tworzony od nowa.
    c) **LS_COLORS –** przechowuje informacje o ustawieniu kolorów przy poleceniu ls dla katalogów, ukrytych plików itd..
    d) **PATH –** przechowuje ścieżki oddzielone separatorami w których ma szukać programów jak jakiś będziemy chcieli uruchomić, np. ls.
    e) **PS1 –** przechowuje informacje o tym jak ma wyglądać znak zachęty czyli to co masz przed każdym poleceniem jak piszesz w terminalu np. [magda@orfi]$
    f) **PWD –** przechowuje bieżącą lokalizację
    g) **HOSTNAME –** przechowuje nazwę hosta.
    h) **EDITOR –** zawiera nazwę domyślnego edytora tekstu

18. Co robi zmienna specjalna $?

Odp. Zwraca kod po wykonaniu polecenia, tzn. jeśli zwróci 0, będzie to oznaczało że polecenie zostało prawidłowo wykonane, jakakolwiek inna liczba oznaczać będzie nieudane wykonanie polecenia. Możemy ją wykorzystać w instrukcjach warunkowych albo żeby sobie sprawdzić to zrobić np. mkdir omega i napisać echo $? i zobaczyć jaki kod zwróci a potem napisać mkdurrrr omega i napisać echo $? i także zobaczyć jaki kod zwróci, najlepiej to zrozumieć w praniu ☺

19. Zmienna specjalna = zmienna systemowa.

Prawda/Fałsz

Odp. Prawda. Zmienne specjalne nazywamy też zmiennymi systemowymi.

20. Czy my(użytkownicy) możemy definiować wartości zmiennych specjalnych? Wybierz właściwe:

Tak/Nie , możemy też/tylko je czytać.

Odp. Nie, możemy tylko je czytać.

21. Wymień dostępne znaki cytowania w powłoce i omów zasadę działania każdego z nich. (4 znaki)

Odp. Dostępne znaki cytowania:

<> akcent `

Wymusza aby to co znajduje się miedzy znakami akcentu zostało zinterpretowane jako polecenie np. echo `date +%Y` czyli to spowoduje, że uruchomimy polecenie, które zwróci nam obecny rok a nie napisze na ekran date +%Y…

<> cudzysłów „

Cudzysłów działa w sposób, że traktuje wszystko jak napis/tekst z wyjątkiem: $, drugiego cudzysłowu i akcentu `. Popróbuj sobie w powłoce, że łatwiej to ogarnąć i zapamiętać.

<> apostrof ‘

Apostrof działa w sposób, że traktuje wszystko jako tekst.

<> backslash \

Backslash działa w sposób, że traktuje następny znak jako tekst np. echo \; wypisze na ekran średnik. Sprawdź jeśli chcesz, żeby się upewnić.

22. **Napisz polecenie za pomocą którego do zmiennej data przypiszesz obecny rok i miesiąc w formacie: RRRR-MM. (musisz przypomnieć sobie polecenie date)**
    Odp.
    data=`date +%Y-%m` <= wykorzystując znak cytowania akcent
    lub
    data=$(date +%Y-%m) <= zwróć uwagę, że tylko jeden nawias

23. **Wykonaj operację matematyczną w powłoce: x*x mając x=4. Skorzystaj z jednego z dostępnych sposobów interpretacji wyrażeń arytmetycznych.**
    Odp.
    Sposób1: let „x=x*x” <= wykorzystując let
    Sposób2: x=$((x*x)) <= wykorzystując nawiasy!!

24. **Czy polecenie *echo Ten tekst;Jest;Widoczny* wyświetli wszystko na ekranie(terminalu)? Jeśli tak to dlaczego? Jeśli nie to jak to rozwiązać?**
    Odp. Te polecenie nie wyświetli średnikow ponieważ uważa je za istotne znaki najpewniej i nie interpretuje ich jako tekstu co bardzo byśmy chcieli. Aby to rozwiązać musimy zastosować znak cytowania np. \ (backslash) aby średnik był interpretowany jako tekst czyli będzie to wyglądało tak dla przykładu: echo Ten tekst\;Jest\;Widoczny ale moglibyśmy skorzytać też z apostrofu, cudzysłowu itd..

    Ale taka sytuacja na marginesie żeby dobrze zrozumieć tą sprawę i być przygotowanym na wszystko. Skoro już o to było pytanie: chcemy takie polecenie wykonać: echo „;;Ten;;te;k;st;;jest napisany w $PWD” to jak optymalnie i szybko zrobić żeby było wypisane tak jak chcemy??? – oczywiście skorzystamy ze znaku cytowania cudzysłów. Musisz pamiętać, że cudzysłów wszystko interpretuje jako tekstu oprócz: $, drugiego cudzysłowu i akcentu czyli ` a więc odpowiedź byłaby taka:
    echo „;;Ten;;te;k;st;;jest napisany w $”

25. **Zdefiniuj słowo skrypt i opisz krótko jak działa.**
    Odp. skrypt – plik tekstowy, który zawiera polecenia powłoki i wszystkie narzędzia programistyczne powłoki(praca na zmiennych, dostarcza instrukcji if, petli itd..)

26. **Dokończ zdanie: Instrukcja if kończy się komendą: ……**
    Odp. Każda instrukcja if, którą napiszemy musi kończyć się komendą zamykającą fi!

27. Wybierz właściwą odpowiedź. Warunek if podajemy następująco:

a) [wyrażenie]
b) [wyrażenie]
c) [wyrażenie]
d) (wyrażenie)

Odp. c) ponieważ musimy pamiętać że w wyrażeniu muszą być odstępy(spacje) przed i po wyrażeniu!! Jest to absolutnie konieczne aby nasz skrypt działał prawidłowo.

28. Podaj ważniejsze wyrażenia: (np. –r, -x, -w, -d) i co robią?

Odp. Jedne z ważniejszych wyrażeń:
-e prawda jeśli plik istnieje
-f prawda jeśli plik istnieje i jest zwykłym plikiem
-l prawda jeśli plik istnieje i jest dowiązaniem symbolicznym
-r prawda jeśli plik istnieje i można go czytać
-s prawda jeśli plik istnieje i ma rozmiar większy od 0
-u prawda jeśli plik istnieje i ma ustawiony bit set-user-id
-w prawda jeśli plik istnieje i można do niego pisać
-x prawda jeśli plik istnieje i można go wykonać
plik1 –nt plik2 (nt – newer than), prawda jeśli plik1 jest nowszy niż plik2
plik1 –ot plik2 (ot – older than), prawda jeśli plik1 jest starszy niż plik2
plik1 –ef plik2 prawda jeśli plik1 i plik2 mają te same numery i-węzła.

Przykłady stosujące te wyrażenia dla przećwiczenia:

a) Wyrażenie, które sprawdza czy mamy prawo r do pliku.
   if [-r nazwa_pliku];then….
b) Wyrażenie, które sprawdza czy plik dane jest starszy niż plik dokumenty_IO
   if [dane –ot dokumenty_IO];then….
c) Wyrażenie, które sprawdza czy podany plik jest plikiem zwykłym
   read –p „Podaj nazwe pliku: „ plik
   if [-f $plik];then……. (zwróć uwagę na $, bez $ powłoka by myślała, że ma sprawdzić czy plik o nazwie plik jest plikem zwykłym, rozumiesz?)

29. Wybierz właściwą odpowiedź i uzasadnij. Chcemy porównać dwa napisy czy są takie same: tomek i atomek, w wyrażeniu zapiszemy zatem:

a) [tomek=atomek] ,na zmiennych [$x=$y]
b) [tomek = atomek], na zmiennych [$x = $y]
c) [tomek –eq atomek], na zmiennych [$x –eq $y]
d) [tomek –n atomek] , na zmiennych [$x –n $y]

Odp. b), do porównywania dwóch ciągów służy nam znak = i musimy pamiętać o odstępach pomiędzy dwoma porównywanymi słowami czy tam zmiennymi jeśli porównujemy zmienne... Jeśli nie zrobimy odstępów to przecież byłoby to przypisanie

prawda? tomek=atomek oznaczałoby, że do zmiennej tomek wrzucamy napis atomek! Trzeba na to uważać!!

30. Wybierz właściwą odpowiedź i uzasadnij. Chcemy porównać dwie liczby czy są takie same: 5 i 6, w wyrażeniu zapiszemy zatem:

a) [5 –eq 6]

b) [5 –ne 6]

c) Oba sposoby są prawidłowe

Odp. c), -eq sprawdza czy liczby są sobie równe a więc porównujemy czy są takie same natomiast –ne sprawdza czy liczby są różne więc także porównujemy je….

31. Jaka jest różnica między: [x=3] a [x = 3]?

Odp. x=3 oznacza przypisanie 3 do zmiennej programowej x, natomiast x = 3 oznacza porównanie napisów x i 3! Trzeba na to uważać, żeby samemu nie popełnić błędów! Aby na to nie dać się nabrać, albo nie popełnić błędu trzeba troche popisać skryptów samemu ☺

32. Jaka jest różnica między: echo $rok a echo rok?

Odp. echo $rok spowoduje, że na ekran zostanie wypisana wartość, którą przechowuje zmienna rok(a jeśli zmienna rok nie przechowuje żadnej wartości to wtedy będzie pusta linia na ekranie – sprawdź aby się upewnić!!). Natomiast echo rok spowoduje wypisanie na ekran napisu rok. Czujesz różnicę? Jak nie to mówić.

33. Uzupełnij miejsca ???. Niech x=5, y=4. Jeśli chcemy porównywać liczby to jak sprawdzamy:

a) = [$x -eq $y] -z ang: equal(pol. równe)

b) != [$x -ne $y] -z ang: not equal(pol. nierówne)

c) < [$x -lt $y] -z ang: less than(pol. mniejsze niż)

d) <= [$x -le $y] -z ang: less equal(pol. mniejsze równe)

e) > [$x -gt $y] -z ang: greater than(pol. większe niż)

f) >= [$x -ge $y] -z ang: greater equal(pol. większe równe)

***pytanie*Dlaczego nie możemy użyć np. = tylko musimy stosować –eq?**

Odp. Ponieważ –eq tyczy się porównywania liczb całkowitych natomiast = tyczy się porównywania dwóch ciągów napisów czy są sobie równe!!!

34. Czy dopuszczalne jest takie sformułowanie warunku?

if [$a = $b] && [$a = $c];then

TAK/NIE

Odp. TAK, możemy stosować w ten sposób operator AND czyli && aby zrobić złożony warunek. Czyli oznacza on, że jeśli pierwsze wyrażenie jest prawdziwe i jednocześnie

drugie wyrażenie jest prawdziwe to wchodzimy do środka if. Możemy też zastosować OR || jeśli tego potrzebujemy.

35. Jakie operatory logiczne możemy wykorzystać w wyrażeniach? (wymień 3.)

Odp.

-a (koniunkcja), a od and

-o (alternatywa), o od or

! (negacja)

36. Jak wygląda pierwsza linia w każdym skrypcie?

Odp. #! /bin/bash

37. Zapytaj o imie i przechowaj imie w zmiennej text, napisz to polecenie:

Odp. read –p „Podaj imie: „ text

38. Jakie znasz sposoby uruchamiania skryptów? Wymień i krótko je opisz.

Odp.

<> za pomocą komendy sh nazwa_skryptu

Powoduje, że **tworzona jest kopia powłoki** i w tej kopii wszystko się dzieje, tam są tworzone zmienne lokalne, po wykonaniu skryptu te zmienne lokalne giną! Zatem jak np. zapytaliśmy w skrypcie o imie i przechowaliśmy to w zmiennej informacja to jak zrobimy echo $informacja to nic się nie wyświetli.

<> za pomocą .[spacja]nazwa_skryptu

Uruchamia skrypt w bieżącej powłoce, stąd wszystkie zmiany są widoczne!, np. zapytaliśmy w skrypcie o imie i przechowaliśmy to w zmiennej informacja to jak zrobimy echo $informacja to wyświetli nam to imie. Jak nie wierzysz to sprawdź ☺. Nawet jak wiesz to sprawdź, żeby być pewnym i utrwalić sobie.

<> nadając prawo x i pisząc ./nazwa_skryptu lub dodając ścieżkę do PATH gdzie skrypt się znajduje i wpisanie nazwy skryptu.

Nadajemy sobie atrybut wykonywalności(x) np. chmod u+x powitanie (powitanie to skrypt załóżmy) i teraz uruchamiamy go jak każdy program, jednak jeśli chcemy je wywoływać tak jak np. pwd, ls itd czyli po prostu pisząc nazwe skryptu to musimy pamiętać o tym, że trzeba dodać ścieżkę w której znajduje się nasz skrypt powitanie do zmiennej środowiskowej PATH, która jak pamiętamy zawiera ścieżki w których powłoka ma przeszukiwać skrypty do uruchomienia – jeśli tego nie zrobimy to powłoka zwróci informacje: command not found. Alternatywnym spobobem jest powiedzenie skąd uruchamiamy skrypt czyli ./powitanie , czyli ogólnie ./nazwa_skryptu ale pamietasz że musimy mieć prawo x do tego skryptu!

39. Jaki znak pozwala zrobić komentarz w edytorze tekstu mcedit?

Odp. Znak % (procentu) – jednowierszowy komentarz.

40. Czy drukarka w LINUX jest plikiem?

TAK/NIE

Odp. Tak, wszystko W LINUX jest plikiem, nawet drukarka.

41. Czym jest deskryptor plików?

Odp. Deskryptor pliku – identyfikator pliku wykorzystywany przez system operacyjny. Po wykonaniu operacji otwarcia pliku, deskryptor pliku może być wykorzystywany wielokrotnie prez wywołanie systemowe w operacjach wejścia/wyjścia. Deskryptor pliku jest zwracany przez funkcje systemowe z rodziny open.

Zgodnie z POSIX deskryptor pliku to liczba całkowita, czyli wartość typu int z języka C. Domyślnie każdy proces po uruchomieniu ma otwarte 3 standardowe deskryptory plików.(czyli stdin = 0, stdout = 1, stderr = 2)

42. Jakie wartości liczbowe są przyporządkowane poszczególnym deskryptorom plików: stdin, stdout, stderr i co one przekazują?

Odp.

0 stdin –klawiatura

1 stdout –terminal danych prawidłowych

2 stderr –terminal błędów

43. W jaki sposób przekierować polecenie ls –l do pliku info aby stworzyć ten plik na nowo? Napisz to polecenie.

Odp. ls –l>info , jeden znak > oznacza stworzenie pliku na nowo

44. W jaki sposób przekierować polecenie ls –l do pliku info aby dopisać do tego pliku otrzymane informacje? Napisz to polecenie.

Odp. ls –l>>info , dwa znaki >> oznaczają dopisanie do pliku już istniejącego

45. Czy dane prawidłowe płyną innym strumieniem niż dane nieprawidłowe?

TAK/NIE

Odp. TAK, żeby zrozumieć przekierowania musismy sobie wyobrażić, że dane prawidłowe płyną innym strumieniem niż dane nieprawidłowe, najlepiej to zobaczysz w zadaniu 47, tam przekierowujesz prawidłowe dane do jedengo pliku a nieprawidłowe do innego. Oczywiście nie musisz np. nieprawidłowych danych wrzucać do innego pliku jeśli nie chcesz, możesz np. skorzystać z opcji: 2>/dev/null aby ich nie wyświetlało i nie przeszkadzało w pliku, który sobie tworzysz, dopisujesz do niego itd. itd.

46. W jaki sposób przekierować zawartość pliku info do pliku kopia? Napisz to polecenie. (podpowiedź: chodzi o przepisanie zawartości pliku info do pliku kopia)

Odp. info<kopia , znak < oznacza przekeirowanie jednego pliku do drugiego, jeśli tego jeszcze nie rozumiesz, warto przypomnieć sobie te polecenie co robliśmy na ćwiczeniach z hasłem i spróbuj je jeszcze raz zrobić.

47. Skonstruuj polecenie: Wynik polecenia ls –l /tmp* zapisz w pliku sprawozdanieLS(ale tylko prawdiłowe dane!) natomiast jakiekolwiek błędy zapisz do pliku niechcianeWYNIKI.

Odp. ls –l /tmp 1>sprawozdanieLS 2>niechcianeWYNIKI

Komentarz:

Ta 1 i 2 są właśnie z deskryptorów plików, przypomnijmy, że 1 oznacza terminal danych prawidłowych a 2 oznacza terminal błędów. Pamiętaj, że dane prawidłowe płyną innym strumieniem niż dane prawidłowe, ten przykład właśnie to pokazuje – błędy zapisujemy w pliku B a prawidłowe dane w pliku A.

- przekierowujemy strumień danych prawidłowych 1 do sprawozdanieLS
- przekierowujemy strumień danych nieprawidłowych 2 do niechcianeWYNIKI

Rozszerzenie zadania:

Aby lepiej je zrozumieć, załóżmy, że teraz chcemy zapisać dane prawidłowe do pliku wiadomosciOLX a bledow nie chcemy zapisywac ani wyswietlac na ekranie, skorzystamy zatem z 2>/dev/null , widzisz analogie z tą 2 dlaczego ona tutaj stoi? To jest właśnie deskryptor plików, który nazywamy terminalem błędów!!!, przerzucamy błędy do null czyli niczego tak naprawdę. Zatem mamy:

ls –l /tmp* 1>wiadomosciOLX 2>/dev/null

Jeśli nie rozumiesz tego to koniecznie poćwicz sobie w powłoce a jeśli niezrozumiale odpowiedź jest to mów. ☺

48. Czym jest potok(pipe)? Jak on działa? Co ma wspólnego z potokiem „tee”?

Odp. potok – mechanizm komunikacji międzyprocesowej umożliwiający bezpośrednią wymianę danych między uruchomionymi procesami. Czyli po prostu podłączamy wyjście jednego procesu do wejścia drugiego: czyli: potok łączy stdout jednego procesu z stdin drugiego procesu.

polecenie1|polecenie2 – użycie potoku

w potoku mamy coś takiego jak tee czyli rozgałęzienie potoku, jest to taka bardziej rozszerzona wersja i wygląda to tak: polecenie1|tee plik|polecenie2

49. Czy polecenie: who|wc –l wykorzystuje tzw.potok?

TAK/NIE

***Opisz krótko jak działa powyższe polecenie?**

Odp. TAK te polecenie wykorzystuje potok, powyższe polecenie działa tak, że wynik polecenia who czyli wyświetlenie listy zalogowanych użytkowników wrzucamy jako dane wejściowe do polecenia wc –l, które zliczy liczbę linii i w ten sposób otrzymamy informację ile użytkowników jest akutalnie zalogowanych.

50. W jaki sposób wyświetlić listę aktualnie zalogowanych użytkowników? (3 sposby – wymień je)

Odp. Możemy wykorzystać polecenie who, finger lub users

51. Co robi polecenie wc?

Odp. Polecenie wc nazwa_pliku zlicza kolejno linie, słowa, znaki w podanym pliku

52. Co zrobi polecenie wc –l /etc/passwd?

Odp. Te polecenie zliczy ile jest linii w pliku passwd. Sprawdź w man wc.

53. Co zrobi polecenie wc –w /etc/passwd?

Odp. Te polecenie zliczy ile jest słów w pliku passwd. Sprawdź w man wc.

54. Co robi polecenie head? Ile ma domyślnie ustawione linii?

Odp. Polecenie head czyta domyślnie 10 pierwszych linii pliku, możemy sami ustalić ile linii ma zczytać np. head -3 /etc/passwd przeczyta trzy pierwsze linie pliku passwd. Sprawdź i się upewnij czy rzeczywiście tak jest!

55. Co robi polecenie tail? Ile ma domyślnie ustawione linii?

Odp. Polecenie tail czyta domyślnie 10 ostatnich linii pliku, możemy to sami ustalić ile ostatnich linii ma zczytać np. tail -5 /etc/passwd.

56. Co spowoduje wykonanie polecenia: tail -1 /etc/passwd

Odp. Polecenie te spowoduje wyświetlenie ostatniej linii pliku passwd

57. Co spowoduje wykonanie polecenia: head -5 /etc/passwd

Odp. Polecenie te spowoduje wyświetlenie pierszych pięciu linii pliku passwd

58. Do czego służy polecenie cut?

Odp. Polecenie cut służy do wycinania pól(słów) z rekordu(linii). Możemy wyciąć znaki, fragmenty stosując różne argumenty(odsyłam do man cut), np. –d itd..

59. Co spowoduje wykonanie polecenia:

a) cut –d":" –f1,3 /etc/passwd

Odp. polecenie te spowoduje wycięcie pierwszej i trzeciej 'kolumny' każdego rekordu, oddzielonej separatorem : z pliku passwd. Sprawdź jak działa, żeby ogarnąć co się dzieje.

b) cut –d":" –f1,3-6 /etc/passwd

Odp. polecenie te spowoduje wycięcie pierwszej kolumny i kolumn od trzeciej do szóstej z pliku passwd oddzielając kolumny separatorami : , no bo wiemy, że jak sobie wyświetlimy plik passwd to zobaczymy, że informacje są oddzielone za pomocą dwukropka, więc możemy ładnie policzyć co chcemy wyciąć żeby tylko to wyświetlić np. jak mamy graba:troszke:sobie:strzelam, to jeśli chcesz wyswietlic strzelam to potrzebujesz wyciąć kolumnę nr 4 prawda? Tak sobie liczymy bo mamy te informacje oddzielone separatorami i w naszym poleceniu cut zaznaczyliśmy –d":" czyli że ma wypisywać to co chcemy do momentu napotkania separatora.

UWAGA: do polecenie CUT musimy sobie przypomnieć jeśli nie pamiętamy co np. ls –l wyświetla w danym miejscu, bo jak nie będziemy wiedzieć to jak dostaniemy zadanie aby tylko wypisać z ls –l prawa grupy to nie będziemy wiedzieli które to są ☺

60. Jak wyciąć poleceniem cut 2,3,10 znak z wyniku polecenia ls –l? (wykorzystaj potok!)

Odp. ls –l | cut –c2,3,10

- wiesz, że potok pozwala wynik jednego polecenia dać jako wejście na inne polecenie, w tym przypadku wynik polecenia ls –l przekazujemy dla polecenia cut
- -c pozwala wyciąć znaki, gdyby cie poproszono o wyswietleniu np. tylko uprawnien pozostalych uzytkownikow to możesz policzyc które to sa znaki i wypisać…

61. Co spowoduje wykonanie polecenia: ls –l|sort ?

Odp. Przekazujemy za pomocą potoku wyjście polecenia ls –l dla polecenia sort, które po prostu sortuje otrzymane dane. Najlepiej to sprawdzi samemu.

62. Co robi polecenie uniq?

Odp. Polecenie uniq usuwa powtarzające się linie(można sobie skojarzyć uniq od unique – unikalne). Działa z sort.

63. Utwórz zmienną ile, która pamięta ilu jest użytkowników zalogowanych w systemie(musisz wykorzystać potok bo masz wynik jednego polecenia ogarnąć innym poleceniem).

Odp. ile=`who|wc –l`

Analizując:
- pamiętamy, że who zwraca liczbe zalogowanych użytkowników
- pamiętamy, że wc zlicza liczbę linii, słów, znaków. Natomiast wc –l zlicza tylko liczbe linii czego idzie się domyślić (-l jest skrótem od –lines). Zatem skoro polecenie who zwraca nam zalogowanych użytkowników w postaci jeden użytkownik na wiersz czyli tak dla przykładu:

Uzytkownik1 14512 host@orfi.uwm.pl
Uzytkownik2 12352 host@orfi.uwm.pl
root root host@orfi.uwm.pl
....

To możemy za pomocą sprawdzenia liczby linii zliczyć ilu użytkowników jest zalogowanych, tak? Cwane to nie? Zauważ, że jak wykonasz pojedynczo te polecenia to właśnie wynik polecenia who dajemy na wejście polecenia wc –l , jakże to wygodne nie? Ten potok jest bardzooo pomocny. ☺

ZATEM DO PRZEĆWICZENIA:
- umiejętność tworzenia skryptów
- umiejętność tworzenia zmiennych
- umiejętność używania polecenia date
- umiejętność stosowania polecenia cut, i innych poleceń które się pojawiły
- umiejętność korzystania z potoku
- umiejętność korzystania z instrukcji warunkowych

Itd…itd…itd..