SYSTEMY OPERACYJNE – NOTATKA

1. **Katalog domowy oznaczamy symbolem**: ~
2. W LINUX rozróżniana jest wielkość liter! Plik **historia** a **Historia** to dwa różne pliki!
3. . w nazwie pliku oznacza, że jest on ukryty(zwykle ukryte są pliki konfiguracyjne..)
4. LINUX nie interpretuje rozszerzeń, nie potrzebuje ich!
5. ls – listuje pliki (bez ukrytych, bez szczegółów)
6. ls –l listuje pliki (bez ukrytych, ze szczegółami)
7. ls –al Listuje pliki (ze szczegółami, +dodatkowo pokazuje ukryte pliki)
8. **Uwaga**: ls –l możemy użyć aby wyświetlić szczegóły tylko jednego pliku np. ls –l info (gdzie info to plik). Może się przydać gdy np. chcemy wyświetlić na ekran właściciela pliku, którego podamy nazwę za pomocą skryptu.
9. Do ls możemy stosować metaznaki np. ls –l ??? (listuje pliki szczegółowo, których nazwa składa się z 3 znaków)
10. ls –l /bin/nap* (listuje pliki szczegółowo, z katalogu bin, których nazwa zaczyna się na nap)
11. pwd – pokazuje bieżącą lokalizację
12. **rodzaje plików:**

- zwykły
d katalog
c urządzenia o dostępie znakowym
b urządzenia o dostępie blokowym e
l linki dołączenia

13. Uprawnienia podstawowe: r(czytanie), w(pisanie), x(wykonywanie)
14. d**rwx**r-xr-x **2** WMIIUWM\141080 WMIIUWM\domain users 4096 mar 17 13:35 cw3

d – typ pliku(katalog)
2 oznacza liczbę dowiązań twardych(linków twardych)
pierwsze trzy **rwx** – prawa właściciela(user)
drugie trzy r-x – prawa grupy (group)
ostatnie trzy r-x - prawa pozostałych użytkowników(others)
WMIIUWM\141080 – właściciel pliku
WMIIUWM\domain users – właściciel grupowy
4096 – rozmiar pliku
mar 17 13:35 – ostatnia modyfikacja
cw3 – nazwa pliku/katalogu itd..

15. **mkdir** nazwa_katalogu – tworzy katalog o podanej nazwie
16. rm –r nazwa_pliku/nazwa_katalogu – usuwa katalog/plik (może być z zawartością)
17. rmdir nazwa_katalogu – usuwa katalog, jednak ten katalog musi być pusty.
18. **metaznaki w nazwach plików**: * dowolny ciąg znaków, ? jeden znak – nie może być pusty, [listaZnakow] – jeden z dowolnych znaków, [a-z] – od do znaku
    przykład: ls –l /bin/a??[0-9]*(wylistuje pliki, których nazwa zaczyna się na a, mają dwa znaki dowolne a potem cyfrę z przedziału od 0 do 9 a potem to już dowolny ciąg znaków)
19. **home** – katalogi domowe użytkowników(tam użytkownik ma swój katalog i ma tam uprawnienia)
20. **prawo r dla katalogu** pozwala robić np. ls

21. **prawo w dla katalogu** wszelkie zmiany katalogu np. dodanie pliku, usuniecie pliku z katalogu, zmiana nazwy pliku, który jest w katalogu, edycja pliku itd.
22. **prawo x dla katalogu** pozwala użyć polecenia np. cd
23. **katalog bin** (od binaries) – przechowuje programy które możemy uruchomić np. ls
24. **polecenia zewnętrzne** – programy które znajdują się w głównej mierze w katalogu bin
25. **polecenia wewnętrzne** – polecenia wbudowane w powłokę
26. type - sprawdzić możemy czy polecenie wbudowane czy nie np.type cd
27. alias – inna nazwa dla sekwencji poleceń
28. ls –l /bin – odwołanie bezwzględne(podaję pełną ścieżkę) – zadziała zawsze!!
29. ls –l bin – zadziała tylko gdy jestem w odpowiednim miejscu!
30. cp kopiowanie plików **cp [opcje] co gdzie** np. skopiować plik /etc/passwd do ~/kolok
    odpowiedź: cp /etc/passwd ~/kolok
31. **katalog /etc** zawiera pliki konfiguracyjne
32. **UWAGA na taką sytuacje!**
    cp /etc/passwd ~/dane – skopiuje plik passwd do folderu dane, nazwa dalej będzie passwd
    cp /etc/passwd dane – skopiuje plik passwd do folderu w którym jesteśmy ale będzie miał on nazwe dane – **tak nie chcemy!!**
33. **Oglądanie zawartości pliku:**
    cat – wyświetla wszystko
    more – tylko w dół można przeglądać
    less – można przeglądać w dół i w górę strzałkami
34. **Plik passwd** – lista kont założonych na serwerze, mamy w nim zapisane dane w taki sposób:
    nazwaKonta:x:UID:GID:opis:katalog domowy:powłoka
    x – kiedyś tu było hasło
    UID(User identity) – identyfikator użytkownika
    GID(Group identity) – identyfikator grupy podstawowej
35. **Plik group** – lista grup założonych na serwerze, mamy w nim zapisane dane w taki sposób:
    nazwaGrupy:x:GID:[lista uzytkowników]
    np. sys:x:3:root, bin, adm
36. **Polecenie id** pokazuje uid, gid i grupy do których należysz, a dla przykładu: id root pokaże uid , gid i grupy do których należy root itd..
37. **Polecenie groups** –pokaże grupy do których należysz, a dla przykładu polecenie groups root pokaże grupy do których należy root
38. **Plik shadow** – plik z hasłami, są tam zaszyfrowane hasła
39. **chmod** – zmiana uprawnień do pliku
    symbolicznie: należy pamiętać, że
    g – grupa(group)
    o – pozostali(others)
    u – użytkownik,właściciel(user)
    a – wszyscy(all)
    przykładowo: chmod g+rw,u+rwx,o+r plik
    (nadajemy grupie prawa r,w właścicielowi r,w,x pozostałym r)

    UWAGA, w trybie symbolicznym mamy +(nadanie praw), -(odebranie praw), =(ustawienie praw TYLKO TYCH, które za jego pomocą nadajemy)

ZNAK = TYLKO TE UPRAWNIENIA NADAJE KTÓRE DAMY, a jeśli sa pozostale to je usunie, mamy np. *chmod g=x passwd* (no to grupa będzie miała tylko x do passwd, r i w znikną jeśli były).

przykład2 ze znakiem = :
przed poleceniem wykonuje polecenie ls –l passwd i otrzymuje, że na tę chwilę są takie uprawnienia do tego pliku: **rwx rwx r-x**
pisząc polecenie:
chmod a=rw passwd
(czyli do pliku passwd ustawiamy wszystkim opcje r i w)
Spowoduje, że uprawnienia do pliku passwd będą teraz takie: **rw rw rw**

chmod w trybie numerycznym:
Chcesz nadać do pliku prawa rw- r-- ---,piszesz postać binarną 110 100 000, przerabiasz ją na postać dziesiętną: 640 i piszemy polecenie:
chmod 640 passwd
Wada chmod numerycznego? - Musimy ustawiać wszystkie prawa! W trybie symbolicznym możemy nadać/odebrać/ustawić prawa dla określonej grupy np. tylko właścicielowi, tylko grupie, pozostałym albo wszystkim jednocześnie(czyli all), albo i grupie i właścicielowi.

40. Polecenie **umask** – ustawia domyślne uprawnienia dla tworzonych plików
41. Polecenie **mv** – zmiana nazwy lub przeniesienie pliku!
    Np. mv info nowa_nazwa (zmiana nazwy)
    mv info ~/katalog (przeniesienie pliku info do ~/katalog)
42. **i-węzeł, i-node, index-node**: struktury opisujące pliki w systemie które zawierają wszelkie informacje zwiazane z plikiem ale bez danych plików i jego nazwy!
43. **Dowiązanie twarde(link twardy)** – referencja wskazująca na zawartość pliku lub katalogu. Dla systemu operacyjnego jest dodatkową nazwą dla pliku – plik z n dowiązaniami ma n nazw. Więc jeśli będziemy chcieli usunąć plik, który ma dowiązania twarde to musimy usunąć też jego wszelkie dowiązania aby pozbyć się go na dobre.
44. **Dowiązanie symboliczne(link symboliczny)** – wskazuje odwołując sięza pomocą nazwy na dowolny inny plik lub katalog(który może nawet w danej chwili nie istnieć).
45. ln –s **zbior_dok** nazwa_linku_symbolicznego – tworzy do pliku **zbior_dok** link symboliczny
46. ln **zbior_dok** nazwa_linku_twardego – tworzy do pliku **zbior_dok** link twardy
47. **Czasy dostępu do pliku**
    atime – informacja o czasie ostatniego dostępu do danych zawartych w pliku, ls nie wpłynie, odczyt danych wpłynie na atime
    mtime – czas ostatniej zmiany zawartości piku, ls nie wpłynie, odczytanie zawartości pliku nie nie wpłynie, edycja pliku wpłynie na mtime np. dodanie linijki usunięcie linijki
    ctime – ulega zmianie kiedy zmieniają się informacje o pliku, ulega zmianie gdy zmieni się właściciel,grupa pliku, prawa dostępu, liczba dowiązań do pliku i przy jakiekolwiek akcji powodującej zmianę mtime!!
48. Polecenie **stat** nazwaPliku
    Wyświetla status pliku, Device – urządzenie na którym jest zapisany plik, Inode (numer i-węzła), ctime, mtime itd.
49. **Polecenie find**, wyrażenie testujące do przeszukiwania katalogów, podkatalogów itd.

Najczęściej używane atrybuty z nim, pamiętajmy też że można je negować itd:
-name (np. zaczyna się na litere r to mamy –name r*)
-maxdepth 1 (maksymalna głębokość: bez podkatalogów)
-mtime (zmiana zawartości np. ostatnie 30 dni to mamy –mtime -31)
-size (rozmiar pliku np. wiekszy niż 4kilobajty to mamy –size +4k)
-user (właściciel pliku)
-type (typ pliku)

Operatory logiczne w powłoce:
! – not (negacja)
-a –and lub nic – koniunkcja
-or lub -o – alternatywa
!! uwaga: and ma wyższy priorytet niż or!!

50. Polecenie **export nazwa_zmiennej** powoduje, że nazwa_zmiennej staje się zmienną globalną środowiskową.
51. Zmienne służą do przechowywania danych.
52. Powłoka nie wymaga deklarowania typu zmiennej!
53. W powłoce Bash zmienne traktowane są jak napisy!! Np. x=x+3 traktuje jako napis x+3
54. Zmienne są podzielone na **3 kategorie**.
    - **zmienne programowe(użytkownika)**
    zmienne definiowane przez użytkownika, te zmienne są kasowane po wyłączeniu powłoki, definicja zmienne programowej wygląda tak: nazwaZmiennej=wartosc np. x=123 Zapisujemy te zmienne umownie małymi literami.

    - **zmienne środowiskowe(globalne)**
    Używane do przekazywania informacji powłoce i programom uruchamianym pod jej kontrolą. Zapisujemy je umownie WIELKIMI LITRAMI. Zmienne globalne można sprawdzić za pomocą polecenia set.

    - **zmienne specjalne(systemowe)**
    Definiowane przez system, nie można zmieniać ich wartości, przykładowe zmienne specjalne to $# , $?

    Aby interpretować **wyrażenia arytmetyczne** musimy wykorzystać jeden ze sposobów poniżej:
    I) polecenie **let** np.
    x=1
    let „x=x+1”
    let „x=x*3”
    II) $ nawiasy np.
    x=1
    x=$((x*4))

55. Polecenie **read** - wczytywanie do zmiennej, stosujemy często z atrybutem **–p**
    np. read –p „Podaj nazwe pliku: „ nazwa
56. Aby usunąć zmienną musimy zastosować polecenie unset, np. **unset** x
57. **readonly** ustawia zmienną tylko do odczytu, **nie można jej zmienić ani usunąć**. Sama znikinie po wyłączeniu powłoki. np. readonly x=5 lub jeśli wcześniej zdefiniowaliśmy zmienną x to readonly x wystarczy!!
58. **Polecenie set oraz env pozwala zobaczyć zmienne środowiskowe.**
59. Przykładowe zmienne środowiskowe, które trzeba ogarniać(zobaczysz je poleceniem set/env):
    - **LOGNAME** – nazwa logowania
    - **HOSTNAME** – nazwa hosta
    - **HOME** – katalog domowy
    - **PWD** – bieżąca lokalizacja
    - **HISTFILE** – plik historii poleceń (możemy dzięki niemu używać strzałki do odtworzenia poleceń)
    - **HISTSIZE** – rozmiar pliku liczony w wierszach (gdy przekroczymy wartość, kasowane są wiersze i od nowa)
    - **LS_COLORS** – ustawienie kolorów dla polecenia LS
    - **PATH** – **ścieżki przeszukiwań** oddzielone dwukropkami ścieżki w systemie plików, gdy chcemy uruchomić program to przeszukuje zmienną PATH czy program znajduje się w katalogu pierwszym, potem kolejnym itd. Jak nie znajdzie to zwraca błąd.
    echo $PATH|grep „$HOME” - sprawdzenie czy katalog domowy jest w scieżce przeszukiwań
    PATH=$PATH:sciezka <- dodanie swojej sciezki do PATH
    - **PS1** – definiuje znak zachęty
    - **EDITOR** – nazwa domyślnego edytora tekstu

60. **$?** Zwraca kod odnośnie poprzednio wykonanego polecenia, jeśli zwróci 0 to oznacza, że poprzednie polecenie wykonało się prawidłowo, jakakolwiek inna wartość oznacza, że poprzednie polecenie nie wykonało się prawidłowo.

61. **Znaki cytowania** – wyłączają bądź wymuszają interpretacjęznaków specjalnych w zależności od tego co chcemy osiągnąć. Dysponujemy następującymi znakami cytowania:
    - apostrof ‘ ‘ - wszystko traktuje jako napisy
    - cudzysłów „ „ – wszystko traktuje jako napisy **prócz $, akcentu` i drugiego cudzysłowu** „
    - akcent `` - wymusza interpretacje tekstu jako polecenia np. ile= `who|wc –l`
    -backslash – następny znak jest traktowany jako napis np. \;

62. Jak zapisać datę do zmiennej?
    ile=`date +%d-%m-%Y`
63. Jak wypisać godzinę/minuty w echo?
    echo `date +%H:%M`
64. Na zmienna miesiac podstaw miesiac z polecenia data
    miesiac=`date +%m`
    w razie problemów odsyłam do mana, man date!

65. **Zapamiętajmy**, że średnik w echo się nie wypisuje np. echo ;;;;dupa wypisze na terminal dupa więc żeby były widoczne, trzeba zastosować jakiś znak cytowania.

66. **Skrypt** – plikt tekstowy, który zawiera polecenia powłoki i wszystkie narzędzia programistyczne powłoki.

67. **Wyrażenia dotyczące plików**(do instrukcji warunkowej if, elfi)
    -d plik istnieje i jest katalogiem
    -e plik istnieje(exist)
    -f plik istnieje i jest plikiem zwykłym
    -l plik istnieje i jest dowiązaniem symbolicznym
    -r plik istnieje i można go czytać
    -s plik istnieje i ma rozmiar większy od zera
    -u plik istnieje i ma ustawiony bit set-user-id
    -w plik istnieje i można do niego pisać
    -x plik istnieje i można go wykonać(execute)
    plik1 –nt plik2 prawda jeśli plik1 jest nowszy niż plik2
    plik1 –ot plik2 prawda jeśli plik1 jest starszy niż plik2
    plik1 –ef plik2 prawda jeśli plik1 i plik2 mają te same numery i-węzła

68. **Porównywanie napisów**(łańcuchów)
    -z ciąg – prawda jeśli ciąg ma długość równą zero (z od zero)
    -n ciąg – prawda jeśli ciąg ma długość większą od zera (n od not zero)
    ciąg1 = ciąg2 prawda jeśli ciągi są jednakowe **(UWAGA NA SPACJE!!)**
    ciąg1 != ciąg2 prawda jeśli ciągi są różne **(UWAGA NA SPACJE!!)**

69. **Porównywanie liczbowe**:
    -eq od equal (oznacza =)
    -ne od not equal (oznacza !=)
    -lt od less than (oznacza <)
    -le od less equal (oznacza <=)
    -gt od greater than (oznacza >)
    -ge od greater equal (oznacza >=)

70. W **case** wzorce mogą używać metaznaków i znaku |

71. Pierwsza linia skryptu: **#! /bin/bash**

72. **Sposoby uruchomienia skryptu:**
    I) sh nazwa_skryptu
    wykonywany w **podpowłoce**(czyli kopii powłoki)
    II) ./nazwa_skryptu

Wykonywany w **podpowłoce**, aby zadziałał trzeba najpierw nadać atrybut x dla właściciela, czyli chmod u+x nazwa_skryptu. Jeśli chcemy odpalić skrypt podając tylko nazwę skryptu to dodatkowo musielibyśmy albo wrzucić nasz skrypt do jednej ze ścieżek przeszukiwań w PATH lub dodać ścieżkę w której się skrypt znajduje do ścieżki PATH.

III) .[spacja]nazwa_skryptu

Wykonywany w **bieżącej powłoce** stąd wszystkie zmiany są widoczne, np. mamy skrypt ze zmienna imie i nadamy przez takie uruchomienie skryptu do niej imie np. maks to jak potem napiszemy echo $imie to rzeczywiscie będzie to co napisalismy do skryptu czyli maks.

73. Jak dodać do zmiennej środowiskowej **PATH kolejną ścieżkę przeszukiwań**?
    PATH=$PATH:$HOME/bin
    Czyli $PATH to stara wartość ścieżki path a po dwukropku podaliśmy $HOME czyli katalog domowy, dodaliśmy go do ścieżki przeszukiwań!
    przykład2:
    PATH=$PATH:$PWD
    Do zmiennej PATH dodaje kolejną ścieżkę przeszukiwań – w tym przypadu ścieżka w której obecnie się znajduję.

74. Co to za plik **/.bash_profile**
    plik konfiguracyjny, który jest wykonywany jak się logujemy, sprawdza czy istnieje plik zwykły bashrc, jeśli tak to go uruchamia.

75. Co to za plik **/.bashrc**
    plik w którym możemy dopisać swoje rzeczy, które chcemy by się wykonywały przy logowaniu.

76. **Deskryptor pliku** to identyfikator pliku wykorzystywany przez SO. 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.

77. Trzy standardowe deskryptory plików:

0 – stdin – klawiatura
1 – stdout – dane prawidłowe
2 – stderr – dane błędne

78. Przekierowania:

> oznacza stworzenie pliku od nowa! (nadpisywanie)
>> oznacza dopisanie do istniejącego pliku!(jeśli pliku nie ma to go tworzy, jak jest - dopisuje)
< oznacza przekierowanie danych z pliku zamiast z klawiatury

79. Proste przekierowania(przykłady jak to działa):

echo Super Duper>tekst (stworzy plik tekst w który będzie napis Super Duper)
echo Bajera>tekst (stworzy plik tekst od nowa, będzie tam tylko napis Bajera)
echo LubiePSI>>tekst (dopisze do pliku tekst LubiePSI i będą tam dwa napisy: Bajera i LubiePSI)
1 linia Bajera

2 linia LubiePSI
read –p „Podaj imie i nazwisko osoby która znasz” zmienna < tekst
wniosek: do zmiennej zmienna zostanie przypisana pierwsza linia z pliku tekst czyli Bajera zamiast czekać na tekst z klawiatury od użytkownika.

80. **Możemy przekierować dane prawidłowe oddzielnie i dane błędne oddzielnie(**pamiętamy że dane prawidłowe płyną innym strumieniem niż dane nieprawidłowe) jak chcemy np.
ls –l /tmp/* 2>/dev/null (błędy kierujemy na null żeby się nie wyświetliły)
ls –l /tmp/* 1>wiadomosci 2>bledy (bledy kierujemy do pliku bledy, dane prawidlowe do pliku wiadomosci)

81. **(Dodatek)**Ciekawa zagrywka do skryptów jest taka, że nie chce wyświetlić np. wyniku grep? Tylko sprawdzić czy poprawnie się wykonał? Wtedy możesz zrobić tak:
set|grep „$tekst” 1>/dev/null
if [$? –eq 0];
itd..
fi
przekierowanie danych prawdiłowych na null nie wpłynie na zwracany wynik polecenia $?

82. Potok(pipe) – mechanizm komunikacji międzyprocesowej który umożliwa bezpośrednią wymianę danych między procesami **– a po ludzku wersja** : możemy wynik jednego polecenia wrzucić na wejście innego polecenia, daje to dużo możliwości:
Schemat potoku: polecenie1|polecenie2|polecenie3|…itd
np. who|wc –l
who – wyświetla zalogowanych użytkowników
wc –l zliczy liczbę linii(wierszy)

83. **tee** – rozgałęźienie potoku (do pliku i na wejście następnego polecenia)
polecenie1|tee plik|polecenie2 np.
set|tee historia|cut –c5-

84. **who,finger,users** – lista aktualnie zalogowanych użytkowników.
85. **wc** – zlicza liczbę linii, słów, znaków
86. **wc –l** - zlicza liczbę linii itd…
87. **head** – czyta początek pliku (domyślnie 10 linii, można samemu ustawić ile linii ma wyświetlić)
88. **tail** – czyta koniec pliku (domyślnie 10 linii, można ustawić ile linii ma wyświetlić)
np. tail -1 /etc/passwd (ostatnia jedna linia z pliku passwd)

89. polecenie **cut** – wycinanie pól/słów
często stosujemy z –d (ogranicznikiem)
#!# ogranicznikiem może być jakikolwiek znak, który oddziela informacje otrzymane w powłoce.
mamy np. dama:lubi:pieski i chcemy wyświetlić tylko pieski to zastosujemy ogranicznik dwukropek bo między każdą informacją jest dwukropek:

echo dama:lubi:pieski|cut –d":" –f3

lub stosujemy z –c (znakiem)
echo dama:lubi:pieski|cut –c1-5 (Wyświetli od 1 do 5 znaku)

90. **sort** – sortuje alfanumerycznie lub numerycznie
91. **uniq** – lista bez powtórzeń, najpierw trzeba użyć sort aby móc użyć uniq!!
92. **Przykładowe polecenia z sort i uniq**:
    wyswietlamy zalogowanych uzytkownikow bez powtorzen dzieki temu!
    who | cut -d" " -f1|sort|uniq

    a teraz wyswietlić duplikaty chcemy (pokazuje zalogowanych więcej niż raz)
    who | cut -d" " -f1|sort|uniq -d

    a teraz ile razy kto jest zalogowany (pokazuje ile razy zalogowany kto jest)
    who | cut -d" " -f1|sort|uniq -c

    a teraz tylko raz zalogowani
    who | cut -d" " -f1|sort|uniq –u (-u od unique)

    gdy zapomnimy opcji uniq, to man uniq!

93. **find** możemy rozszerzyć o opcję –exec aby wykonać dla każdego znalezionego pliku/katalogu jakieś rzeczy.
    -**exec** polecenie uruchamia polecenie dla każdego odnalezionego pliku
    -**ok** polecenie ; tak jak exec ale dodatkowo pyta o potwierdzenie przed wykonaniem polecenia.

    Uwaga!!!
    łańcuch **'{}'** jest podmieniany na obecnie przetwarzana nazwę pliku
    Pamiętajmy o konieczności cytowania znaków specjalnych!

    Przykład działania:
    find /tmp -type f -user magda 2>/dev/null -exec cp -v '{}' ~/cw7 \;
    opis powyższego polecenia:
    wyszukuje pliki zwykłe których właścicielem jest magda w ścieżce /tmp, błędy przekierowuje do /dev/null a dla każdego znalezionego pliku jest wykonywane polecenie cp czyli kopiujemy każdy znaleziony plik do ~/cw7 , opcja –c powoduje wyświetlanie wyników na ekran (pomocne aby zobaczyć czy działa nasze polecenie)

94. Bit lepki (sticky bit), prawo t

    - **sticky bit ustawiony dla katalogu**
    Powoduje ze kazdy ma do niego dostep, moze zapisywac w nim swoje pliki, edytowac je ale nie moze usunac tych plikow ktore tam dorzucil ani usunac katalogu.

- **sticky bit ustawiony dla pliku wykonywalnego**
Przechowuje w pamięci operacyjnej plik po zakończeniu jego działania dzięki czemu przy następnym włączeniu uruchamia się szybciej.

Jak nadajemy bit lepki?
chmod 1000 test
chmod 1251 test
chmod 1621 test
lub
chmod o+t test
(przypomnienie o-others –pozostali)
czyli pierwszy bit od lewej gdy jest 1 to oznacza bit lepki !!

95. Bit specjalny, **prawo „s”** setuid, setgid, możemy ten bit nadać właścicielowi lub grupie
u+s – ustawienie prawa s dla właściciela
g+s – ustawienie prawa dla grupy

prawo „s” ustawione do plików wykonywalnych(setuid czyli właściciela):
powoduje uruchomienie danego programu na prawach właściciela pliku a nie na użytkownika który go uruchamia

nadajemy je w ten sposób:
chmod u+s nazwa_pliku
lub tak
chmod 4000 nazwa_pliku
chmod 4215 nazwa_pliku
chmod 4621 nazwa_pliku itd…

prawo „s” ustawione w trybie setgid(czyli grupy):
powoduje uruchomienie programu na prawach grupy posiadacza pliku a nie na użytkownika który go uruchamia

nadajemy je w ten sposób:
chmod g+s nazwa_pliku
lub tak
chmod 2000 nazwa_pliku
chmod 2412 nazwa_pliku
chmod 2652 nazwa_pliku itd..

***Uwaga:** chmod 6213 nazwa_pliku spowoduje nadanie prawa s i dla właściciela i dla grupy!
Gdyby dać chmod 7212 to 7 oznacza: 4-s dla właściciela, 2-s dla grupy, 1-bit lepki czyli 4+2+1=7 z pierwszego bitu

96. **SetUID** zajmie miejsce w informacji o prawach wykonania dla właściciela
97. **SetGID** zajmie miejsce w informacji o prawach wykonania dla grupy

98. **Sticky bit(t)** –zajmie miejsce w informacji o prawach innych użytkowników(others)
Podpunkty 94,95,96 znajdziemy oczywiście w poleceniu ls –l w miejscach gdzie są prawa

99. Przykład: **rws rwS –x**
Uwaga! Jeżeli **jest wielka litera S to oznacza to, że pod spodem nie ma prawa x**!!!
czyli zapisalibyśmy w postaci binarnej to tak: 6761, rozumiesz? Pierwsza 6 to 4 i 2 bo mamy s dla właściciela(SUID) i prawo s dla grupy(GUID)

100. Prawo s zobaczymy robiąc np. polecenie: ls –l/usr/bin/passwd

101. Jak znaleźć pliki które mają ustawione prawo s –setuid(dla właściciela)?

find /usr/bin –perm -4000
Lub
find /usr/bin –perm –u=s

spójrz:
-perm +prawa któryś z bitów praw jest ustawiony dla plików
-perm –prawa wszystkie bity prawa są ustawione dla pliku. (dokładnie te prawa które damy)
(znajdziesz to w man find)

Pliki do których oba prawa s i właściciela(SUID) i grupy(GUID) jest:
find /usr/bin –perm -6000 (zauważ że przed 6 jest -, oznacza że dokładnie ma być 6 czyli 4 i 2

Pliki do których jest prawo s dla właściciela(SUID) lub dla grupy(GUID):
find /usr/bin –perm +6000 (zauważ, + oznacza 4 lub 2)
sprawdź sobie w bashu findem stosując –exec czyli:
find /usr/bin –perm +6000 –exec ls –l '{}' \;

---

grep – polecenie przeszukujące wnętrza plików!!

Np. grep „root" /etc/passwd (wyszuka linie w pliku passwd gdzie znajduje się słowo root)

Wyrażenia regularne do grep'a:
^ -początek linii
$ -koniec linii
. jeden znak, lub brak znaku
* dowolną ilość powtórzeń znaku poprzedzającego (kompletnie inne znaczenie niż w METAZNAKACH)
.* dowolną ilość powtórzeń dowolnego znaku

jak damy: a* to np. wyszuka. aaa , aa, a itd..

wyszukać w /etc/passwd linie, które zawieraja słowo "root"
na poczatku linii
odp. grep "^root" /etc/passwd

wyszukać w /etc/passwd linie, które zawieraja na końcu linii /bin/bash
odp. grep "/bin/bash$" /etc/passwd

wyszukać w /etc/passwd linie, które nie zawierają na końcu linii /bin/bash
(szukamy tej opcji w man, man grep)
odp. grep -v "/bin/bash$" /etc/passwd
-v odwraca sens dopasowania, wybiera linie niepasujące do tego co szukamy!!!

wyszukać w /etc/passwd linie, które zaczynają się słowami: kad a potem bez znaczenia jakie znaki to będą.
odp. grep „kad.*” /etc/passwd

wyszukać w katalogu /etc pliki (nazwy plików), które zawierają słowo PATH
uwaga, uwzględnić wielkość liter
odp. grep -l "PATH" /etc/passwd
od wyświetlania nazw plików służy polecenie –l

policzyć ile jest plików które nie mają na końcu /bin/bash
odp. grep -vc "/bin/bash$" /etc/passwd
-c od count zlicza liczbe wystąpień
-v odwraca sens dopasowania

wyszukać i ignorować wielkie litery bo mamy biN a nie bin jak normalnie ma ten katalog:
odp.grep -i "/biN/bash$" /etc/passwd
do ignorowania wielkości znaków służy –i

znaleźć pliki(same nazwy plików) które np. zawierają polecenie export
odp.grep -l "export " /etc/* 2>/dev/null
od wyświetlenia samych nazw plików służy polecenie -l

sprawdzić, czy na ścieżce przeszukiwań jest wasz katalog domowy
(katalog domowy ze zmiennej HOME bierzemy)
PATH - ścieżka przesukiwań
odp. echo $PATH|grep "$HOME"
wynik echa przekujezmy na grepa

polecenie **tr** – pracuje na pojedynczych znakach, tr od translacji(translate), zamienia znaki

ls -l ~ (~ - katalog domowy)
chcemy by wystąpienia małego x zostały zamienione na duży X
odp. ls -l ~ | tr "x" "X"

albo chcemy wszystko na dużych literach żeby było
odp. ls -l ~ | tr "a-z" "A-Z"

albo chcemy wszystkie małe litery zamienić na znak @
odp. ls -l ~ | tr "a-z" "@"

chcemy zaszyfrować liczby i np od 0-9 żeby było kropki
odp. ls -l ~ | tr "0-9" "."

!! kolejne zastosowanie tr:
- redukcja/zamiana nadmiernej ilości znaków tego samego typu np. spacji na jeden znak.
dla polecenia who mamy tam np. dużo spacji a my chcemy zrobić,
żeby były tylko jedne spacje pomiędzy informacjami które zwraca
polecenie who, zrobimy to tak:
who | tr –s " " ";"

-s oznacza: powtarzające się znaki są zamieniane na jeden znak

Dzięki temu możemy wyciąć z polecenia who użyteczne informacje za
pomoca polecenia cut!! Jakie to dobree

zadanie:
Z wyniku polecenia who | tr -s " " wyświetlić tylko nazwę logowania
i godzinę zalogowania!!

who | tr –s " " ";"|cut -d";" -f1,4

*Uwaga, pomiędzy –s a „ „ jest odstęp!
Jeśli nie zrobimy tego odstępu to wyszkoczy błąd,
Warto o tym pamiętać.

---

for

Wykonywane są instrukcje dla każdej wartości zwracanej przez zmienną

```
for i in lista; do
  instrukcja1
  instrukcja2
done
```

i – nazwa zmiennej
pętla wykona się tyle razy ile jest elementów w lista!!!

do pętli for, while i wgl do jakiekolwiek innej dostępnej pętli możemy użyć
continue – instrukcja, która przy wywołaniu „skipuje(skip od pomijać)" dalszą część pętli i po prostu wraca do początku pętli(zwiększa nasze „i" i robi dalej instrukcje)
break – instrukcja, która po wywołaniu wychodzi z pętli, natychmiast!

UWAGA do zapamietania!

➔ *Break i continue działają tylko i wyłącznie w pętlach!*

Poniżej przyklady zastosowania continue i break.

Przykład zastosowania **continue**:
Mamy za zadanie wyszukać pliki zwykłe w katalogu domowym i wypisać je na ekran a jeśli to nie jest plik zwykły to napisać o tym, że nie jest to plk zwykły.

```
for i in ~/*;do
        if [ -f $i ];then
                echo Znaleziono: $i (plik zwykły)
                continue
        fi
        echo $i – nie jest plikiem zwykłym
done
```

Przykład zastosowania **break**:
Mamy za zadanie przeszkać katalog domowy, jeśli znajdzie katalog – wypisuje go i kończy działanie, w przeciwnym wypadku wypisuje na ekran nazwe pliku, który znalazł.

```
for i in ~/*;do
        if [ -d $i ];do
                echo $i to katalog! Koncze dzialanie!
                break
        fi
echo znaleziono: $i
done
```

przykład1(szczególnie ważny do zapamiętania, użyteczny!):

Przy każdej iteracji zmienna $i będzie przyjmować jako wartość kolejne nazwy plików z katalogu /bin

```
lp=1
for i in /bin/*;do
  echo ${lp}. $i
lp=$((lp+1))
done
```

przykład2:

Wyświetla zawartość listy czyli [ala ola ula]

```
i=1
for zm in ala ola ula;do
        echo $i. $zm
        i=$((i+1))
done
```

przykład3

Wyświetla listę zalogowanych użytkowników

```
i=1
for zm in `users`;do
        echo $i. $zm
        i=$((i+1))
done
```

przykład4

Postać przydatna pętli for do analizowania plików, tu otrzymamy liste wszystkich plików z katalogu domowego

```
i=1
for zm in ~/*;do
        echo $i. $zm
        i=$((i+1))
done
```

- ~/* oznacza wszystkie pliki pod katalogiem domowym

Mogłoby być np. ~/ab* (lista składa się z plików/katalogów które zaczynają się na ab)

możemy np. zrobić, żeby wyświetlił tylko pliki które zaczynają się na litere n wtedy by było:

~~~~~~~~~~~~~~~~~~~~

```
i=1
for zm in ~/n*;do
        echo $i. $zm
        i=$((i+1))
done
```

Budowa pętli while

```
while warunek;do
  instrukcja1
  ....
done
```

przykładowy program:

```
ile=3
while [ $ile -gt 0 ];do
        echo $ile
        ile=$((ile-1))
done
```
~~~~~~~~~~~~~~~~~~~~