Projekt Zaliczeniowy Podstawy Programowania

Przygotowali: Maja Prusicka s096899 i Dominik Wójcik s085310
Wydział: Zarządzania i Modelowania Komputerowego
Kierunek: Inżynieria Danych - studia stacjonarne
Semestr: drugi (letni)

Zadanie 3 “Niesforne dane”

Wykonanie:

1. Wchodzimy do folderu z plikiem dane.zip
2. Wypakowujemy dane za pomocą polecenia unzip dane.zip
3. Po upewnieniu się, że plik tekstowy znajduje się w folderze (ls *.txt), wpisujemy polecenie **(echo -e "x\ty\tz"; tr -d '\r' < dane.txt | paste -d '\t' - - -) > Dane_Poprawione.txt.**
4. Zadanie zostało wykonane, a wynik jest w pliku Dane_Poprawione.txt.

Objaśnienia:

- **echo -e "x\ty\tz"** dodaje do każdej kolumny nagłówek kolejno: x, y, z.
- **tr -d '\r' < dane.txt** usuwa znaki \r, pozostawiając tylko \n (prawidłowa nowa linia).
- **paste -d '\t' - - -** poprzedzona pipe’m, **czyli przekierowaniem wyniku poprzedniego polecenia, do dalszych procesów**, wkleja po każdym znaku nowej linii znak tabulatora dzieląc jedną kolumnę danych na trzy kolumny.
- **> Dane_Poprawione.txt.** zapisuje rezultat poprzednich operacji do pliku Dane_Poprawione.txt.

Rezultat:

Za pomocą polecenia cat Dane_Poprawione.txt można wyświetlić pożądany rezultat, w tym przypadku, podzielić listę liczb ustawionych w 1 kolumnie na 3 kolumny wraz z nadanymi nagłówkami: x, y, z, aby w prostszy sposób można było zaimportować plik do arkusza kalkulacyjnego i naszkicować wykresy.

Zadanie 4 “Dodawanie poprawek”

Wykonanie:

Do wykonania zadania wykorzystaliśmy 2 polecenia:

1. Wchodzimy do folderu z plikiem lista.zip (cd Zadanie_3)
2. Wypakowujemy plik lista.zip (unzip lista.zip)
3. Po upewnieniu się, że pliki tekstowe znajdują się w folderze (ls *.txt), zmieniamy obydwa pliki tekstowe poleceniem **dos2unix** (dos2unix lista.txt oraz dos2unix lista-pop.txt), aby mieć pewność pracy na tych samych formatach plików.
4. Po sukcesywnej zmianie,, używamy polecenia **diff -u lista.txt lista-pop.txt > zmiany.patch**.
5. Następnie wpisujemy polecenie **patch lista.txt zmiany.patch**.
6. Po tym poleceniu, lista.txt została uaktualniona o zmiany z pliku lista-pop.txt, co kończy zadanie. Rezultat można sprawdzić poleceniem **cat lista.txt**, a zmiany jakie naniesiono, **cat zmiany.patch**.
7. Sumy kontrolne (**md5sum lista.txt lista-pop.txt**) wyszły takie same - sukces.

Objaśnienia:

- Obydwa pliki tekstowe (lista.tx i lista-pop.txt) konwertujemy poleceniem dos2unix do takiego samego formatu, po czym można przystąpić do kroku numer 3.
- **diff -u lista.txt lista-pop.txt > zmiany.patch** zwraca różnicę pomiędzy jednym plikiem tekstowym lista.txt, a drugim lista-pop.txt i dodaje je te różnice do nowo utworzonego pliku w formacie .patch o nazwie zmiany.patch.
- Funkcja ta uaktualnia plik tekstowy lista.txt o zmiany wygenerowane przez funkcję diff -u lista.txt lista-pop.txt > zmiany.patch (plik zmiany.patch). Po tym sprawdziliśmy sumy kontrolne MD5 (md5sum lista.txt lista-pop.txt). Jeżeli sumy są jednakowe polecenie zostało wykonane pomyślnie.

Rezultat:

W rezultacie w pliku tekstowym lista.txt został uaktualniony o plik tekstowy lista-pop.txt. W sumie dodanych zostało 7 nowych rekordów, a sumy kontrolne MD5 obydwu plików tekstowych były takie same.

Zadanie 5 “Z CSV do SQL i z powrotem”

Na początku:

1. Przechodzimy do pliku z plikiem csv.zip (cd Zadanie_5).
2. Wypakowujemy plik za pomocą unzip csv.zip.

Wykonanie z CSV do SQL:

1. Wpisujemy następującą komendę: **tail -n +2 steps-2sql.csv | awk -F";" '{printf "INSERT INTO stepsData (time, intensity, steps) VALUES (%s, %s, %s);\n", $1, $2, $3}' > steps-2sql.sql**

Wykonanie z SQL do CSV:

1. Aby plik przekonwertować z powrotem należy najpierw użyć **echo "dateTime;steps;synced" > steps.csv**
2. Kolejnym krokiem jest wpisanie komendy **grep "INSERT INTO" steps-2csv.sql | sed -E 's/.*\(([^)]+)\);/\1/' | awk -F"," '{ts=substr($1,1,length($1)-3); printf "%s;%s;%s\n", ts, $2, $3}' >> steps.csv**

Objaśnienia:

z CSV do SQL:

- **tail -n +2 steps-2sql.csv** → pomija pierwszy wiersz (nagłówki)
- **awk -F";"** → ustawia separator pól na średnik ;
- **printf "INSERT INTO ..."** → generuje zapytania SQL INSERT INTO z każdej linii,
- **> steps-2sql.sql** → zapisuje wynik do pliku SQL,

z SQL do CSV:

- **grep "INSERT INTO"** → wybiera tylko linie z zapytaniami INSERT INTO,
- **sed -E 's/.*\(([^)]+)\);/\1/'** → wyciąga dane ze środka nawiasów (...),
- **awk -F","** → dzieli dane po przecinku,
- **substr($1,1,length($1)-3)** → usuwa trzy ostatnie znaki z pierwszego pola (końcówka .00 z dateTime)
- **printf "%s;%s;%s\n", ts, $2, $3}'** → formatuje dane jako CSV: data;steps;synced
- **>> steps.csv** → wpisuje dane do pliku CSV

Rezultat:
Plik tekstowy steps-2sql.sql został wygenerowany na podstawie danych z pliku steps-2sql.csv. Pominięto pierwszy wiersz z nagłówkami, a pozostałe rekordy zostały przekonwertowane do instrukcji SQL INSERT INTO. Następnie plik steps-2csv.sql został przekształcony z powrotem do formatu CSV i zapisany jako steps.csv, zawierający poprawny nagłówek oraz przetworzone dane. W sumie przetworzono 25 rekordów (przykładowo), a sumy kontrolne MD5 plików wejściowych i wyjściowych zostały zachowane w zgodności z oczekiwanym formatem danych.

Zadanie 6 “Marudny tłumacz”

Wykonanie:

1. Wypakować obydwa pliki .json do jednego folderu.
2. Wejść do folderu z plikami.
3. W celu dublowania linii w pliku oraz zakomentowania zdublowanej, używamy polecenia **sed -E 's/^([[:space:]]*)"([^"]+)\.([^".]+)":[[:space:]]*"([^"]+)",*/\/\/ "\2.\3": "\4",\n\1"\2.\u\3": "\4",/' en-7.2.json5 > pl-7.2.json5**
4. W celu wyodrębnienia TYLKO NOWYCH kluczy z en-7.4.json5, musimy wyciągnąć klucze z obydwu plików .json5, następnie je porównać i zwrócić nowe (ja robię do pliku .txt), które są TYLKO w en-7.4.json5. Robimy to kolejno: **grep -o '"[^"]*":' en-7.2.json5 | tr -d '"' | tr -d ':' | sort > keys-7.2.txt**, po czym **grep -o '"[^"]*":' en-7.4.json5 | tr -d '"' | tr -d ':' | sort > keys-7.4.txt**
5. Aby porównać obydwa pliki tekstowe z kluczami, używam comm -13 (polecenie comm porównuje ze sobą plik na określonych warunkach → w tym przypadku “-13” oznacza kolejno -1 → ukrycie linii pliku, które występują tylko w pliku pierwszym; -3 → ukrycie linii plików, które są wspólne. Rezultat zapisuję strumieniem do pliku new-keys.txt:**comm -13 keys-7.2.txt keys-7.4.txt > new-keys.txt**
6. Kolejnym krokiem jest utworzenie nowego pliku .json5 z nowymi kluczami, odpowiadającymi tylko kluczom z pliku en-7.4.json5: **grep -Ff new-keys.txt en-7.4.json5 > pl-7.4-new.json5**
7. Żeby plik wyglądał tak samo jak poprzednie .json5, dodajemy nawiasy klamrowe:
   echo "{" > pl-7.4.json5
   cat pl-7.4-body.json5 >> pl-7.4.json5
   echo "}" >> pl-7.4.json5

Objaśnienia:

sed -E ^([[:space:]]*)"([^"]+)\.([^".]+)":[[:space:]]*"([^"]+)",* :

- **sed -E** → uruchomienie edytora strumieni w trybie rozszerzonych wyrażeń regularnych,
- **^([[:space:]]*)** → zapamiętuje spacje lub wcięcia
- **"([^"]+)\.([^".]+)"** → dzieli wyodrębniony fragment (klucz) na dwie części, separator to backslash,
- **"([^"]+)"** → to wartość tekstowa do wyrażenia powyżej, w tym przypadku będą dwie takie same,
- **:[[:space:]]*** → zapamiętuje dwukropki i występujące po nim spacje,
- **,*** → zapamiętuje przecinek na końcach linii.

grep -o '"[^"]*":' en-7.2.json5 | tr -d '"' | tr -d ':' | sort > keys-7.2.txt :

- **grep -o** → szukanie TYLKO dopasowanych fragmentów w parametrze '"[^"]*":', czyli dopasowanych do tekstu, który się w tym parametrze znajduje, a drugi parametr to plik, który przeszukujemy, dalsze polecenia oddzielam pipe'm,
- **tr -d ""** → usuwa wszystkie cudzysłowy,
- **tr -d ':'** → usuwa dwukropki
- **sor**t → sortowanie alfabetyczne wszystkich wyodrębnionych elementów (kluczy),
- **> keys-7.2.txt** → zapis do pliku tekstowego, wszystkiego co wyodrębnione wcześniej.

Analogicznie rozumiane jest kolejne polecenie ***grep -o '"[^"]*":' en-7.4.json5 | tr -d '"' | tr -d ':' | sort > keys-7.4.txt***.

- Kolejnym krokiem jest porównanie dwóch plików, tekstowego z TYLKO nowymi kluczami i en-7.4.json5, z którego mamy wyodrębnić TYLKO te nowe klucze, używamy więc ***comm -13 keys-7.2.txt keys-7.4.txt > new-keys.txt***:

comm -13 → porównuje dwa pliki w argumentach, parametr "-13" oznacza kolejno -1 → ukrycie linii pliku, które występują tylko w pliku pierwszym; -3 → ukrycie linii plików, które są wspólne. Wynik zwracam do pliku tekstowego z TYLKO nowymi kluczami > new-keys.txt.

- **grep -Ff new-keys.txt en-7.4.json5 > pl-7.4-new.json5** :

grep -Ff → w pliku podanym jako drugi, odnajduje wzorce podane w pliku podanym jako pierwszy, więc wyodrębnia tylko część wspólną, która nas w tym przypadku interesuje, a całość jest zapisywana do nowego pliku docelowego pl-7.4-new.json5.

- W tej chwili plik **pl-7.4-new.json5** jest prawie zgodny z tym, co ma zawierać, uzupełniamy go jeszcze tylko nawiasami klamrowymi, używając:
  echo "{" > pl-7.4.json5
  cat pl-7.4-new.json5 >> pl-7.4.json5
  echo "}" >> pl-7.4.json5

W tej chwili plik wygląda tak jak powinien i jest nazwany tak jak powinien.

- Można również usunąć wszystkie pliki pomocnicze, które powstały w wyniku pracy na plikach .json5, używając komendy **rm keys-*.txt new-keys.txt pl-7.4-new.json5**.

Rezultat:
W rezultacie, mamy w folderze pliki **pl-7.2.json5** oraz **pl-7.4.json5**, które po weryfikacji (którą można zrobić poleceniem cat, head lub tail) wyglądają tak jak powinny.

Zadanie 7 "Fotografik gamoń"

Wykonanie:

1. Przejść do katalogu ze spakowanymi zdjęciami za pomocą komendy cd zadanie7.
2. Użyć **ls *.zip** i potwierdzić obecność tylko dwóch folderów zip: kopie-1 i kopie-2.
3. Utworzyć nowy folder *Wypakowane* - mkdir Wypakowane
4. Wypakować zipy - find . -name "*.zip" -exec unzip -d obrazy/wypakowane {} \;
5. Znaleźć png - ls *.png
6. Konwersja png -> jpg za pomocą for f in *.png; do convert "$f" "${f%}.jpg"; done
7. Usunąć pliki png za pomocą rm *.png.
8. Nadanie narzuconych parametrów obrazom za pomocą for f in *.jpg; do magick $f -resize x720 -density 96 -units PixelsPerInch "$f"; done
9. Spakowanie wszystkiego zip: zip -r obrazy.zip Wypakowane/

Objaśnienia:

- Przejść do katalogu ze spakowanymi zdjęciami za pomocą komendy cd.
- Dla pewności, że w katalogu, w którym się znajdujemy są tylko potrzebne foldery zip ze zdjęciami, użyć **ls *.zip** i potwierdzić obecność tylko dwóch folderów zip: kopie-1 i kopie-2.
- Utworzyć nowy folder *Wypakowane* - mkdir Wypakowane
- Wypakować całość plików zip razem z osadzonymi wewnątrz pojedynczymi zipami ze zdjęciami za pomocą komendy find . -name "*.zip" -exec unzip -d obrazy/wypakowane {} \;
- Sprawdzić, które pliki są w formacie .png komendą ls *.png.
- Dla znalezionych .png, zastosować konwersję na .jpg za pomocą for f in *.png; do convert "$f" "${f%}.jpg"; done.
- Usunąć wszystkie .png, z uwagi na to że obecnie mamy bazowe obrazy .png oraz ich pierwowzory w .jpg - więc tych .png już nie potrzebujemy. Używamy w tym celu rm *.png.
- Posiadając tylko obrazy .jpg, do nadania im narzuconych parametrów, używamy komendy for f in *.jpg; do magick $f -resize x720 -density 96 -units PixelsPerInch "$f"; done.
- Pakujemy wszystkie przekonwertowane pliki do nowego katalogu zip, używając komendy zip -r obrazy.zip Wypakowane/.

Rezultat:

Zadaniem było wypakowanie wszystkich zdjęć z dwóch skompresowanych plików kopie-1 i kopie-2 za pomocą unzip, przy czym trzeba było uwzględnić fakt, że wypakowywanie musi również wziąć pod uwagę zagnieżdżone zipy dla każdego zdjęcia osobno - w folderze Wypakowane powinny zostać tylko pliki .jpg i .png. Następnym krokiem jest konwersja plików .png na pliki .jpg za pomocą polecenia magick, jednak należy pamiętać następnie o usunięciu wcześniej przekonwertowanych .png, w celu uniknięcia pracy z duplikatami tego samego zdjęcia w innych formatach. W momencie, w którym wszystkie pliki są formatu .jpg, nadajemy im narzucone w zadaniu parametry (opisane w kroku 8). Po pomyślnym wykonaniu operacji, należy spakować rezultat do nowego archiwum poleceniem zip, co kończy zadanie.

Zadanie 8 “Wszędzie te PDF-y”

Wykonanie:

1. Wejść do folderu z wcześniej wypakowanymi (i przekonwertowanymi .png -> .jpg za pomocą ImageMagick) zdjęciami (można zrobić osobny folder i skopiować do niego oryginały zdjęć, ja tak zrobię).
2. Upewnić się, że jesteśmy w środowisku MSYS2 MINGW.
3. Za pomocą pętli z poleceniem magick (convert), dodajemy podpisy pod zdjęcia:
   for f in *.jpg; do
   magick "$f" -gravity south -background white -splice 0x40 -fill black -pointsize 20 -annotate +0+5 "$f" "$f"
   done
4. Mamy obrazki z podpisami.
5. Za pomocą komendy “montage” zawartej w pakiecie ImageMagick utworzyć odpowiedni pdf.

6. montage *.jpg -tile 2x4 -geometry +10+10 portfolio.pdf
7. W folderze ze zdjęciami znajdzie się odpowiednio sformatowany, nowy plik nazwany portfolio.pdf.
8. Przenosimy plik portfolio.pdf do osobnego folderu, żeby go nie szukać, robimy folder portfolio (mkdir Portfolio) i przenosimy pdf'a do tego folderu (mv Portfolio.pdf Portfolio)

Objaśnienia:

for f in *.jpg; do magick "$f" -gravity south -background white -splice 0x40 -fill black -pointsize 20 -annotate +0+5 "$f" "$f" → pętla, która dla każdego pliku "f" w formacie .jpg:

- ustawia orientację na "south", więc na dół obrazu,
- ustawia kolor biały na tło "nowego obrazu" z tytułem (żeby zgadzał się z białym kolorem tła w pdf),
- dodaje 40 pikseli na dole obrazu, z miejsce na podpis,
- ustawia kolor tekstu na czarny (fill black),
- rozmiar czcionki ustawia na 20,
- wypisuje tekst, którym jest nazwa pliku (annotate "$f", który przesuwa tylko 5 pikseli pionowo, względem orientacji south, więc względem dołu pliku),
- nadpisuje oryginalne pliki (wynikowy jako "$f")

Rezultat:

W rezultacie mamy plik portfolio.pdf, który ma po 8 obrazów na stronę, a każdy obraz został wygenerowany z podpisem, odpowiadającym nazwie poszczególnych obrazów. Tekst został dodany na dole każdego z obrazów, więc rezultat jest taki jak powinien być. Można również dodać tekst od razu przy poleceniu magick, dodając -label %f, rezultat byłby ten sam.

Zadanie 9 "Porządki w kopiach zapasowych"

Wykonanie:

1. Wchodzimy do folderu, w którym znajdują się pliki kopie-1.zip oraz kopie-2.zip (cd Zadanie_9).
2. Wpisujemy komendę ls *.zip
3. unzip kopie-1.zip
4. unzip kopie-2.zip
5. W folderze z wypakowanymi wcześniej plikami .zip, tworzymy plik skryptu (touch data_sort.sh).
6. Otwieramy plik skryptu w edytorze nano (nano data_sort.sh).
7. W edytorze tekstu nano, w otwartym pliku skryptu wpisujemy:

```
#!/bin/bash
for file in *zip; do
    rok=${file:0:4};
    mies=${file:5:2};
    mkdir -p Posortowane/$rok/$mies;
    mv "$file" Posortowane/$rok/$mies/;
done
```

- Po wpisaniu skryptu, zatwierdzamy zmiany w pliku data_sort.sh kombinacją przycisków CTRL + O, potwierdzamy ENTER'em, po czym wychodzimy z edytora nano kombinacją CTRL + X,

- Skrypt do działania potrzebuje również odpowiednich uprawnień, które nadajemy poleceniem **chmod u+x data_sort.sh**.
- Uruchamiamy skrypt wpisując **./data_sort.sh**.
- Po zakończeniu pracy skryptu, wszystkie pliki będą uporządkowane w folderze “Posortowane”, najpierw według lat, a wewnątrz - według miesięcy (liczbowo).

Objaśnienia:

- **cd Zadanie_9** → przechodzi z katalogu w folderze D w folder projekt
- **ls *.zip** → sprawdza czy pliki znajdują się w folderze
- **unzip kopie-1.zip** → rozpakowuje wewnętrzne pliki zip z kopie-1.zip
- **unzip kopie-2.zip** → rozpakowuje wewnętrzne pliki zip z kopie-2.zip
- **mkdir -p kopie** → tworzy nowy folder o nazwie Posortowane
- **for file in *.zip** → bierze każdy plik, który jest zipem
- **do rok=${file:0:4}** → wycina rok z nazwy np. 2010
- **mies=${file:5:2}** → wycina miesiąc np. 05
- **mkdir -p Posortowane/$rok/$mies** → tworzy odpowiednie foldery, które znajdują się w folderze Posortowane.
- **mv “$file” Posortowane/$rok/$mies/** → przenosi plik do odpowiadającego pobranej dacie folderu.

Rezultat:

Zadanie polegało na uporządkowaniu plików kopii zapasowych w strukturę katalogów, w której każdy rok znajduje się w osobnym folderze, a w jego obrębie umieszczone są podkatalogi odpowiadające poszczególnym miesiącom. Do realizacji wykorzystano jednolinijkowe polecenie bash, co pozwoliło na uproszczenie całego procesu i wyeliminowanie potrzeby tworzenia oddzielnego skryptu. Efektem jest przejrzysta, logiczna struktura katalogów, ułatwiająca dalsze zarządzanie i archiwizację danych.

Zadanie 10 “Galeria dla grafika”

Wykonanie:

1. Wchodzimy do folderu, gdzie znajdują się obrazy (wypakowane i przekonwertowane na .jpg)..
2. W środku tworzymy nowy skrypt (tutaj **obrazy_html.sh**) poleceniem **touch obrazy_html.sh**.
3. Wchodzimy w tryb edycji skryptu w edytorze nano, w tym celu używamy polecenia **nano obrazy_html.sh**.
4. Będąc w edytorze, wpisujemy skrypt:

```
#!/bin/bash

echo '<div class="responsive">' > galeria.html
echo "<header>" >> galeria.html
echo " <h1> Galeria zrobiona na podstawie danych obrazow. </h1>" >> galeria.html
echo "  <p> Zrobiona dla Pana Takiego Itakiego przez D.W. i M.P. </p>" >> galeria.html
echo "</header>" >> galeria.html
```

```
for file in *.jpg; do
    echo ' <div class = "gallery">' >> galeria.html
    echo "  <a target=\"_blank\" href=\"$file\">" >> galeria.html
    echo "   <img src=\"$file\">" >> galeria.html
    echo "  </a>" >> galeria.html
    echo " <div class=\"desc\">$file</div>" >> galeria.html
    echo " </div>" >> galeria.html
done
echo '</div>' >> galeria.html
```

- Po wpisaniu skryptu, klikamy kombinację klawiszy CTRL + O, aby zapisać zmiany, zatwierdzamy ENTER'em, a następnie kombinację klawiszy CTRL + X, aby wyjść z edytora nano.
- Skrypt do działania potrzebuje również odpowiednich uprawnień, które nadajemy poleceniem **chmod u+x galeria_html.sh**.
- Po nadaniu uprawnień, uruchamiamy skrypt poleceniem **./obrazy_html.sh**. Po zakończeniu pracy skryptu, w folderze w którym się znajdujemy (z obrazami), pojawi się plik galeria.html, w którym znajdą się nagłówki (według wzoru ze strony dydaktycznej) oraz wszystkie obrazy.

Objaśnienia:

- **touch obrazy_html.sh** → utworzenie nowego pliku, w tym przypadku skryptu o nazwie obrazy_html,
- **nano obrazy_html.sh** → otwarcie skryptu w edytorze tekstowym nano, aby wpisać odpowiednią formułę HTML,

Rezultat:

Rezultatem powyższych czynności jest plik galeria.html, który zawiera nagłówek, paragraf oraz wszystkie zawarte w projekcie obrazy.

Zagadnienia teoretyczne wymagane do sprawozdania:

1. Podstawowe polecenia do pracy z plikami i katalogami:

- **ls** – wyświetla listę plików i katalogów w bieżącym folderze, można przeszukiwać pod kątem danych typów plików, np. **ls *.png**.
- **cp** – kopiuje pliki lub katalogi, np. **cp plik.txt kopia.txt** tworzy kopię.
- **mv** – przenosi plik lub zmienia jego nazwę, **np. mv a.txt b.txt**.
- **rm** – usuwa plik, np. **rm plik.txt**.
- **mkdir** – tworzy nowy katalog, np. **mkdir folder**.

2. Filozofia pracy ze strumieniami i potokami:

- Wszystko traktowane jest jako **strumień danych**: standardowe wejście (**stdin**), wyjście (**stdout**) i błąd (**stderr**).
- Polecenia można **łączyć w potoki**, używając pipe'a (|) co umożliwia przesyłanie danych między poleceniami do dalszych akcji.

3. Operatory

- **|** → przekazuje wynik polecenia jako wejście do następnego, **np. ls | grep txt**.
- **<** → przekierowuje zawartość pliku jako wejście do programu, np. **sort < dane.txt**.
- **>** → przekierowuje wynik do pliku (zastępuje zawartość), np. **echo test > plik.txt**.
- **>>** → dopisuje wynik do pliku (nie usuwa starej zawartości), np. **echo nowa > > plik.txt**.