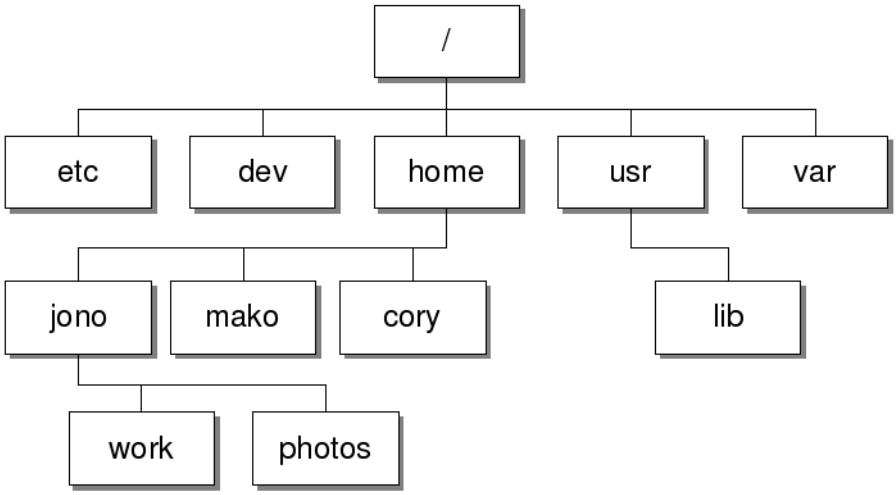
Wstęp do systemu LINUX

18 X – 8 XI 2016

System plików w systemie LINUX



1. System plików zaczyna się od "/" (root).
2. Ścieżki w systemie wyrażane są jako ciąg katalogów, rozdzielanych znakiem "/".
3. Do **każdego** pliku czy katalogu prowadzi ścieżka **bezwzględna** od root'a systemu plików, np. do katalogu z obrazkami użytkownika jono: `/home/jono/photos` natomiast do konkretnego zdjęcia: `/home/jono/photos/wakacje.jpg`
4. W każdym katalogu znajdują się dwa specjalne katalogi: **"."** (= obecny katalog) oraz **".."** (= katalog nadrzędny). Nie mają one odzwierciedlenia na dysku, służą tylko do wygodnego poruszania się po systemie plików w trybie tekstowym (terminala).
5. Katalog domowy można wyrazić za pomocą specjalnego znaku **~**, tym samym zapis `~/photos` jest równoważny `/home/jono/photos`.
6. Ścieżka względna podawana jest w stosunku do aktualnego katalogu, w którym się znajdujemy:

W jakim katalogu jestem	**Ścieżka względna do zdjęcia z wakacji**
/	`home/jono/photos/wakacje.jpg` **LUB** `./home/jono/photos/wakacje.jpg`
/home	`jono/photos/wakacje.jpg` **LUB** `./jono/photos/wakacje.jpg`
/home/jono	`photos/wakacje.jpg` **LUB** `./photos/wakacje.jpg`
/home/jono/photos	`wakacje.jpg` **LUB** `./wakacje.jpg`
/home/jono/photos/z-aparatu	`../wakacje.jpg`
/home/mako	`../jono/photos/wakacje.jpg`
/etc	`../home/jono/photos/wakacje.jpg`
/usr/lib	`../../home/jono/photos/wakacje.jpg`

Terminal

Terminal nazywany jest również shellem lub linią komend, stanowi tekstowy interfejs do obsługi systemu operacyjnego. Nie jest to rzecz specyficzna tylko dla systemów Linuxowych – patrz programy `cmd` czy `powershell` na Windowsa.
Terminala można uruchomić wybierając odpowiednią pozycję z menu systemowego czy też paska szybkiego uruchamiania, lub używając kombinacji klawiszy (zależnej od dystrybucji systemu, często: **Ctrl + Alt + T**). Na systemie Windows można korzystać z programu Cygwin, który emuluje działanie terminala Linuxowego. Aby dotrzeć do systemu plików dostępnego z poziomu terminala, wchodziiy do `C:\Cygwin\StudentXX\`. Polecam otworzyć sobie przeglądarkę plików windowsa i w ten sposób mieć podgląd czy operacje, które wykonujemy mają zamierzony efekt.

Kilka przydatnych informacji:

1. Klawisz **tab** ma przydatną funkcję auto-uzupełniania, która wyjątkowo dobrze sprawdza się przy nawigacji po systemie plików. Przykładowo, jeśli napiszemy `"/home/jono/p"` i wciśniemy klawisz tab, ścieżka zostanie uzupełniona do `"/home/jono/photos"`
2. Klawisze **strzałek góra/dół** pozwalają nawigować po historii wpisywanych komend, jest to przydatne jeśli chcemy wykonać podobną komendę, zmieniając tylko jej fragment.
3. Klawisze **home/end** oraz **strzałki lewo/prawo** z modyfikatorami (**ctrl, alt**) pozwalają nawigować po obecnie wpisywanej komendzie.
4. Wykonywanie komendy można przerwać kombinacją klawiszy **Ctrl + C**

Komendy i znaki specjalne do opanowania:

man

Pozwala uzyskać informacje jak używać poszczególnych programów linii komend. Podajemy nazwę programu jako argument programu **man,** po spacji. Przykładowo, aby wyświetlić manual programu **ls**, wykonujemy**:**
~$ man ls
Aby nawigować, używamy strzałek góra/dół. Aby wyjść z programu, naciskamy klawisz **q**.

pwd

Wypisuje na ekran ścieżkę bezwzględną do katalogu, w którym się obecnie znajdujemy.

```
~$ pwd
/home/student27
```

ls

Służy do kilku rzeczy, ale dla nas na początek będzie istotne listowanie katalogów, czyli wypisywanie listy katalogów i plików znajdujących się w zadanym katalogu. Domyślnie (czyli jeśli nie zostaną podane żadne argumenty) komenda wypisze zawartość obecnego katalogu. Zatem poniższe dwa wywołania są jednoznaczne:

```
~$ ls
~$ ls .
```

Wylistuj zawartości różnych katalogów, używają ścieżek względnych i bezwględnych. Co się stanie jak podamy zwykły plik jako argument wywołania funkcji? Używając programu **man** i własnych obserwacji, zapoznaj się z działaniem opcji **-a, -l, -lh, -R**.

cd

Przechodzi do zadanego katalogu.
Posiłkując się komendami **pwd** oraz **ls** w celu odczytania obecnej ścieżki i zawartości katalogu, wypróbuj poniższe sposoby użycia komendy `cd` i zinterpretuj ich działanie:

```
~$ cd
```

```
~$ cd /usr
~$ cd lib
```

```
~$ cd .
```

```
~$ cd ..
```

mkdir

Tworzy nowy katalog o zadanej nazwie, poprzez ścieżkę względną lub bezwzględną. Aby komenda się powiodła, musi istnieć zadana ścieżka.
Wejdź do swojego katalogu domowego i stwórz tam następującą strukturę katalogów:
`zajecia/2016/kolory`
`zajecia/2016/zwierzeta`
Do czego służy opcja **-p**?

touch

Domyślnie ten program służy do czegoś zupełnie innego (modyfikacja czasu dostępu), ale popularnie używa się go do tworzenia pustych plików.
Stwórz parę pustych plików, postaraj się używać ścieżek względnych i bezwględnych oraz tworzyć pliki znajdując się w różnych katalogach.
`zajecia/2016/kolory/czerwony.txt`
`zajecia/2016/kolory/zielony.txt`
`zajecia/2016/kolory/niebieski.txt`
`zajecia/2016/zwierzeta/kot.txt`
`zajecia/2016/zwierzeta/pies.txt`
`zajecia/2016/zwierzeta/lama.txt`

cat

W założeniu używany do łączenia kilku plików w całość, ale najczęściej do wypisania zawartości pliku na konsolę.
Wypisz zawartość pliku **.bashrc** i innych ukrytych plików znajdujących się w katalogu domowym (użyj odpowiedniej opcji **ls** aby je wyświetlić).

stat

Wyświetla informacje na temat wybranego pliku – jego atrybuty.
Przetestuj działanie tej komendy.

seq

Wyświetla sekwencję kolejnych liczb całkowitych z zadanego przedziału.
Przetestuj działanie tej komendy.

rm

Usuwa zadany plik.
Przetestuj działanie tej komendy, tworząc pusty plik a następnie go usuwając. Przetestuj działanie opcji **-r**, która służy do rekursywnego usuwania katalogów. W tym celu stwórz pliki:
`zajecia/2016/niepotrzebne/plik.txt`
`zajecia/2016/niepotrzebne/inny-plik.txt`
`zajecia/2016/niepotrzebne/bardzo-niepotrzebne/plik.txt`
a następnie usuń cały katalog **niepotrzebne**.

mv

Przesuwa zadany plik/katalog w inne miejsce lub zmienia nazwę (co też jest specyficznym przypadkiem przesunięcia).
Przetestuj działanie tej komendy, w tym celu stwórz następującą strukturę katalogów i plik:
`zajecia/2016/pomylka/ojojoj/plik.txt`
Używając komendy **mv** spraw, aby powyższy plik znalazł się pod następującą ścieżką:
`zajecia/2016/teraz/dobrze/ufff.txt`
(czyli zmień nazwy `pomylka → teraz`, `ojojoj → dobrze`, `plik.txt → ufff.txt`).
Podpowiedź: wymaga to użycia komendy mv kilkukrotnie.
Następnie przenieś plik `ufff.txt` do swojego katalogu domowego i zmień mu nazwę na `plik.txt`.

echo date `` > >>

Wypisuje zadany tekst na konsolę:

```
~$ echo "Dzien dobry"
Dzien dobry
```

Z pozoru nieprzydatna komenda, ale umiejętnie używana jest bardzo pożyteczna. Obrazuje to poniższy przykład. Używa on komendy **date**, która zgodnie z intuicją podaje aktualną datę, oraz **cudzysłowów** typu **backtick**, które powodują że ich zawartość jest wykonywana i wynik wstawiany jest w miejsce komendy.

```
~$ echo "Dzien dobry, dzisiaj mamy `date`"
Dzien dobry, dzisiaj mamy Sat Oct 22 15:06:36 CEST 2016
```

Innym przydatnym znakiem specjalnym jest znak **>**. Pozwala on skierować wyjście programu do wybranego pliku. Wypełnij stworzone przez nas pliki w poniższy sposób, używając metody z **backtickami**.

Plik	Zawartość
`zajecia/2016/kolory/czerwony.txt`	Czerwony jest moim ulubionym kolorem
`zajecia/2016/kolory/zielony.txt`	Jak to pisalem, była godzina <godzina, ale nie pełna data tylko godzina>*
`zajecia/2016/kolory/niebieski.txt`	Jak to pisałem, byłem w katalogu <obecny katalog>*
`zajecia/2016/zwierzeta/kot.txt`	Mój kot jest czerwony, a jak wiadomo <zawartość czerwony.txt>*
`zajecia/2016/zwierzeta/pies.txt`	Plik kot ma następujące atrybuty: <atrybuty pliku kot.txt>*
`zajecia/2016/zwierzeta/lama.txt`	W tym pliku jest zupełnie wszystko: <wszystkie poprzednie pliki sklejone ze soba>*

* użyj odpowiedniej komendy w **backtickach**

$ZMIENNE >> for

Kolejny znak specjalny to **>>**, który działa podobnie do **>**, jednak nie nadpisuje zawartości pliku.
Wypróbuj działanie operatorów **>** oraz **>>** pisząc różne napisy w różnych konfiguracjach do jakiegoś pliku i oceniając efekt za pomocą komendy **cat**.

W linii komend możemy używać zmiennych. Przypisywanie wartości odbywa się operatorem **=** (koniecznie bez spacji!), a odczytywanie za pomocą operatora **$**.

```
~$ ZMIENNA=16
~$ echo "Wartosc zmiennej to $ZMIENNA"
Wartosc zmiennej to 16
```

W linii komend możemy używać pętli. Najprostsza konstrukcja wygląda następująco:

```
~$ for i in `seq 1 6`; do echo "$i"; done
1
2
3
4
5
6
```

i stanowi tutaj zmienną pętli.

Stwórz plik `zajecia/2016/liczby.txt`. Zapisz do niego następującą zawartość:
To jest linia 1.
To jest linia 2.
To jest linia 3.
...
To jest linia 100.

more, less, head, tail

Pozwalają na odczytywanie zawartości plików w różnych trybach.
Przetestuj działanie tych komend na pliku `zajecia/2016/liczby.txt`. W razie potrzeby posiłkuj się manualem. Czym się różnią te komendy? Wyświetl pierwsze 20, a

potem ostatnie 30 linii tego pliku.

cut

Służy do wycinania porcji z każdej linii pliku. Składnia jest nastepująca:

```
~$ cut -d'.' -f2 plik
```

Taki zapis oznacza: weź każdą linię pliku „plik”, podziel ją na fragmenty na znakach „.” i wypisz na ekran drugi fragment każdej linii.
Używając tej komendy, wypisz na ekran same numery linii, które znajdują się i w pliku `zajecia/2016/liczby.txt`. Oczekiwane wyjście:

```
1.
2.
3.
[...]
100.
```

tr

Zamienia wybrane znaki na inne znaki lub usuwa konkretne znaki. Wprowadzimy sobie tutaj nowy znak specjalny: **|** (zwany pipe). Oznacza on, że zamiast wypisywać wynik poprzedniej komendy na konsolę, należy go podać jako wejście do kolejnej komendy. Wtedy **tr** można użyć w następujący sposób:

```
~$ echo "ciag,oddzzielony,przecinkami" | tr ',' ' '
ciag oddzzielony przecinkami
```

Zamieniliśmy znaki przecinków na spacje. Można też po prostu usunąć znaki, opcja **-d**:

```
~$ echo "ciag,oddzzielony,przecinkami" | tr -d ','
ciagoddzzielonyprzecinkami
```

Używając komend **cut** i **tr** oraz znaku specjalnego **|** wypisz zawartość pliku `liczby.txt` w taki sposób, aby wyjście nie zawierało kropek:

```
1
2
3
[...]
100
```

oraz tak, aby zawierało średniki na końcach linii:

```
1;
2;
3;
[...]
100;
```

grep

Pozwala znaleźć linie w tekście, które zawierają zadany tekst. Sposób użycia:

```
~$ grep "napis" plik
```

Wypisze wszystkie linie w pliku „plik”, które zawierają słowo „napis”.
We wzorcu do wyszukiwania można użyć znaków specjalnych `.` oraz `*`. Oznaczają one kolejno „jeden dowolny znak” lub „dowolny ciąg”. Przykładowo, do wzorca: „a.c” pasują ciągi „abc”, „a1c”, „a|c”, „a7c”, natomiast do wzorca „a*c” pasują ciągi „a to nie jest c”, „ablablablac”, ale też „a1c”. Pamiętajmy, najlepiej wzorzec zawsze umieszczać w cudzysłowie, gdyż w innym przypadku komenda może być źle zinterpretowana.
Używając tej komendy, znajdź linie w pliku `liczby.txt`, które zawierają cyfrę 1.
Rozszerz komendę z poprzedniego zadania (**cut + tr**) tak, aby na konsoli pojawiły się tylko liczby podzielne przez 10.

chmod

Pozwala zmienić prawa dostępu do pliku / katalogu. Tutaj przyda się krótkie wprowadzenie do systemu uprawnień w Linuxie.
Każdy plik i katalog ma trzy rodzaje uprawnień:

1) **read (R)** – prawo do czytania pliku lub listowania katalogu
2) **write (W)** – prawo do pisania do pliku
3) **execute (X) –** prawo do wykonywania pliku lub wchodzenia do katalogu

oraz 3 rodzaje użytkowników, których one dotyczą:

1) **user (U) –** właściciel pliku
2) **group (G) –** członek grupy, do której należy plik (to jest teraz mało istotne)
3) **other (O) –** każdy inny użytkownik systemu.

Aby podejrzeć uprawnienia konkretnego pliku, wystarczy wykonać komendę **stat**. Aby zobaczyć uprawnienia wszystkich plików w katalogu, można posłużyć się komendą **ls -l**. Uprawnienia będą wyglądały przykładowo tak:

```
-rw-r--r--
```

Znaczenie poszczególnych znaków jest następujące:

```
suuugggooo
```

s – znak specjalny, np. dla katalogu: „`d`”, dla pliku bez wartości: „`-`”
uuu – uprawnienia dla usera w formacie `rwx`
ggg – uprawnienia dla grupy w formacie `rwx`
ooo – uprawnienia dla innych w formacie `rwx`
Pauza oznacza brak uprawnienia. Przykładowo:

a) `r--` → dana klasa użytkowników ma tylko prawa do odczytu (read)
b) `-wx` → dana klasa użytkowników ma prawa do zapisu (write) i wykonania (execute)
c) `---` → dana klasa użytkowników nie ma żadnych praw

Komenda chmod używana jest w następujący sposób:

```
~$ chmod u+x plik.txt     # daje userowi execute
~$ chmod g-rw plik.txt    # odbiera grupie read i write
~$ chmod uo+rwx plik.txt  # daje userowi innym wszystko
```

```
~$ chmod +r plik.txt      # daje wszystkim read
~$ chmod ugo+r plik.txt   # jw, synonim
```

Ćwiczenia z chmod. Pomiędzy każdymi dwiema instrukcjami używaj **stat** lub **ls -l** aby podejrzeć aktualne uprawnienia pliku.

1) Stwórz `plik.txt`. Zabierz sobie prawa do zapisu, spróbuj coś do niego napisać używając **echo ... >**. Daj z powrotem prawa do zapisu i spróbuj ponownie.
2) Odczytaj `plik.txt`. Zabierz sobie prawa do odczytu i spróbuj ponownie. Przywróć sobie prawa do odczytu.
3) Stwórz `katalog`, a w nim 3 pliki o dowolnych nazwach. Zabierz sobie prawa do odczytu i wykonania na tym katalogu. Spróbuj wylistować zawartość katalogu lub do niego wejść. Przywróć sobie poszczególne prawa i spróbuj ponownie.

Ciekawostka – w systemie Windows system uprawnień jest bardzo podobny, ale użytkownik często nie musi mieć o nim pojęcia aby korzystać z komputera.

Znaki specjalne

W systemie Linux jest pewna grupa znaków specjalnych, które wyrażane są specjalną kombinacją znaków, a nie dosłownie. Jest to potrzebne, gdyż znaki te mają przypisane specjalne funkcje. Przykłady:

Znak	Kod znaku	Wytłumaczenie
`enter`	`\n`	Znak **enter** służy do zatwierdzania wpisanej komendy, dlatego nie możemy użyć go bezpośrednio w tekście. Aby zapisać tekst/komendę ze znakiem nowej linii, używamy odpowiedniego kodu.
`tab`	`\t`	Znak **tab** służy do podpowiadania dalszej części komendy lub ścieżki.
`backslash`	`\\`	Pojedynczy znak **backslash** znaczy dla systemu, że zaraz po nim wystąpi pewien symbol tworzący razem z nim jakiś kod znaku specjalnego. Dlatego aby zapisać pojedynczy znak backslash w tekście, podwajamy go.
`double quote`	`\"`	Znak cudzysłowu rozpoczyna i zakańcza ciąg znaków (string). Aby zamieścić w tekście znak cudzysłowu pomijając tę specjalną funkcję, musimy użyć kodu.
`single quote`	`\'`	Tutaj sytuacja jest prawie taka sama jak wyżej. Różnica jest taka, że tekst w podwójnym cudzysłowie jest interpretowany przez bash, a w pojedynczych jest dokładnie taki jak zapisano (ALE uzględniane są kody znaków). Różne znaki cudzysłowu można mieszać bez używania kodów.

Aby lepiej zrozumieć działanie tych znaków, przetestuj poniższe komendy:

```
~$ echo "pierwsza linia\ndruga linia\n\nostatnia"
~$ echo "tabulator\tsluzy\twcinaniu\ttekstu,\tomnomnom"
~$ echo "to jest znak backslash: \\"
~$ echo "Moj ulubiony film to \"Ojciec Chrzestny\"."
~$ echo "Moj ulubiony film to 'Ojciec Chrzestny'."
```

```
~$ echo 'Moj ulubiony film to \"Ojciec Chrzestny\".'
~$ echo 'Moj ulubiony film to "Ojciec Chrzestny".'
~$ echo 'Dzisiaj mamy `date`.'
~$ echo "Dzisiaj mamy `date`."
~$ echo 'pierwsza linia\ndruga linia\n\nostatnia'
~$ echo 'tabulator\tsluzy\twcinaniu\ttekstu,\tomnomnom'
~$ echo 'to jest znak backslash: \\'
```

Jeśli wydaje Ci się to kompletnie pokręcone, to masz rację :). Warto zapamiętać zasady: podwójny cudzysłów jest interpretowany, pojedynczy nie, zawsze używamy kodów znaków.

sort

Komenda sort pozwala posortować ciągi znaków na wejściu, przyrównując do siebie kolejne linie. Przetestuj działanie komendy:

```
~$ echo '5\n295\n51\n626\n6\n4\n381\n375\n34\n2' | sort
```

Dlaczego wynik programu jest taki, a nie inny? Czy sortowanie zadziałało? Czy da się posortować te ciągi traktując je jako liczby, a nie ciągi znaków?

sed

Bardzo przydatny program pozwalający na manipulację tekstem za pomocą wyrażeń regularnych. My skupimy się na najprostszej funkcjonalności podmiany ciągów znaków. Aby podmienić wystąpienia słowa `wzorzec` w każdej linijce tekstu na `docelowy`, wykonujemy poniższą komendę:

```
~$ echo 'W tym tekscie wzorzec wystepuje\nw dwoch linijkach: wzorzec,
wzorzec.' | sed 's/wzorzec/docelowy/g'
```

Nie wchodząc w szczegóły: literka **`s`** na początku oznacza podmianę, literka **`g`** na końcu oznacza, że w każdej linijce mają zostać podmienione wszystkie wystąpienia wzorca (domyślnie podmieniane jest tylko pierwsze – usuń **`g`** i zobacz efekt).

skrypty w bash

Składnia, której się uczymy na tych zajęciach należy do języka **bash**, który jest standardem w większości shelli Linuxowych. On dostarcza nam te wszystkie konstrukcje typu **`> >> | `` for.`** Programy, których używamy **(ls, echo, cd, ...)** nie są częścią języka bash, ale systemu operacyjnego linux, natomiast zostały stworzone specjalnie do używania w shellu i świetnie współpracują z **bash'em**.
Często zachodzi potrzeba powtarzalnego wykorzystywania jednej skomplikowanej komendy, lub wręcz krótkiego programu w języku **bash**. Moglibyśmy za każdym razem żmudnie wklepywać go do konsoli, ale na szczęście mamy do dyspozycji coś takiego jak skrypty. Podobnie jak w języku python, jest to po prostu plik tekstowy z zapisanym kodem do wykonania.
Tworzymy nasz pierwszy skrypt:

1) Stwórz plik `skrypt.sh` w swoim katalogu domowym. `.sh` to typowe rozszerzenie dla skryptu bashowego, natomiast jest ono tylko dla użytkownika, system traktuje

go jako część nazwy i niezależnie czy dopiszemy tam `.pdf` czy `.png`, plik może być uruchomiony jako skrypt.

2) Otwórz plik `skrypt.sh` w jakimś graficznym edytorze tekstu. Na systemie windows z Cygwinem polecam odnaleźć program **Notepad++**, po czym otworzyć ścieżkę `C:\Cygwin\studentXX\skrypt.sh` do edycji.
3) Wpisz poniższą zawartość do pliku `skrypt.sh` i zapisz go:

```
#!/bin/bash

# Petla, ktora wypisze 10 linijek na ekran.
for i in `seq 1 10`
do
     echo "Licznik petli wynosi $i"
done
echo "Koniec skryptu."
```

Linie rozpoczęte znakiem `#` oznaczają komentarze i są ignorowane podczas wykonania programu. Pierwsza linijka, rozpoczynająca się specjalną kombinacją `#!` podpowiada systemowi, że ma interpretować ten skrypt w języku bash. Zazwyczaj nie jest konieczna, gdyż domyślnym zachowaniem systemu jest właśnie interpretacja w bash, ale jest to dobra praktyka. Widzimy tutaj znaną nam konstrukcję pętli, jednak różni się brakiem średników – dzięki temu że plik jest wielo-linijkowy, bash nie potrzebuje średników aby jednoznacznie zinterpretować konstrukcję pętli.

4) Spróbuj uruchomić nasz skrypt. Aby to zrobić wystarczy podać do niego ścieżkę. Ale uwaga, jeśli znajdujemy się w katalogu nadrzędnym, samo wpisanie komendy `skrypt.sh` nie zadziała – jest to zabezpieczenie przez przypadkowym wykonaniem skryptu. Aby go uruchomić, używamy ścieżki bezwzględnej lub względnej rozpoczętej znakami `./`, czyli: `/home/student28/skrypt.sh` lub `./skrypt.sh` (znajdując się w katalogu domowym).
5) Program nie powinien zadziałać, zamiast tego pojawi się kod błędu. Co należy zrobić aby wykonać program?

Używając znanej już składni przekieruj wynik programu do pliku `wynik.txt`. Upewnij się, że w pliku wylądowała oczekiwana zawartość.

W skryptach możemy używać zmiennych, jest to nawet wskazane. Do zmiennej łatwo jest przypisać wyjście z zadanej komendy, używając backticków. Wklej poniższy skrypt do jakiegoś pliku i uruchom go, testując jego zachowanie:

```
#!/bin/bash

ZMIENNA=`seq 1 10`
echo 'Wynik komendy: echo $ZMIENNA'
echo $ZMIENNA
echo 'Wynik komendy: echo "$ZMIENNA"'
echo "$ZMIENNA"
ZMIENNA=`echo "$ZMIENNA" | tr '\n' ';'`
echo 'Wynik komendy: echo $ZMIENNA'
echo $ZMIENNA
echo 'Wynik komendy: echo "$ZMIENNA"'
echo "$ZMIENNA"
```

Zwróć uwagę na różnicę w zachowaniu programu **echo** dla wielo-linijkowego tekstu w zależności, czy użyjemy cudzysłowu. Jest to kolejna osobliwość basha, do której się należy przyzwyczaić.

W języku bash mamy pewne predefiniowane zmienne. Część z nich pozwala na sprawdzenie argumentów skryptu, które zostały podane przy jego uruchomieniu. Aby lepiej zrozumieć, wklej poniższy program do jakiegoś pliku, przeanalizuj i uruchom go, dopisując argumenty do komendy wywołania:

```
echo "Ten skrypt zostal uruchomiony komenda $0"
echo "Podane argumenty skryptu to: $@"
echo "Ilosc argumentow wynosi: $#"
if [ -z "$1" ]; then
     echo "Skrypt nie ma pierwszego argumentu";
else
     echo "Skrypt ma pierwszy argument";
     if [ "$1" == "-a" ]; then
           echo "Uzyto pierwszego argumentu -a"
     elif [ "$1" == "-b" ]; then
           echo "Uzyto pierwszego argumentu -b"
     else
           echo "Nieznany pierwszy argument: '$1'"
     fi
fi
```

Przykładowe komendy wywołania:

```
~$ ./skrypt.sh

~$ ./skrypt.sh "String w cudzyslowie"

~$ ./skrypt.sh -a

~$ ./skrypt.sh -b cos

~$ ./skrypt.sh -opcja wartosc "String w cudzyslowie"
```

W skrypcie możemy zobaczyć konstrukcję **if**. Nie jest zbyt piękna w języku bash, trzeba się do niej przyzwyczaić. Warto zwrócić uwagę na słowa kluczowe: `if, elif, else, fi`. Bardzo ważne jest zachowanie spacji pomiędzy nawiasami kwadratowymi i wyrażeniami wewnątrz, a także średniki w odpowiednim miejscu.

Kolejnymi przydatnymi operacjami są pętla **while** oraz działania arytmetyczne. Pętla **while** może posłużyć na przykład czytaniu zawartości zmiennej linijka po linijce i przetwarzaniu. Poniżej program, który sumuje liczby od 1 do 10. Przetestuj go:

```
# W ZMIENNEJ znajduje sie lista od 1 do 10, oddzielona zn. nowej linii: \n
ZMIENNA=`seq 1 10`
SUMA=0
# Petla while czyta ZMIENNA linia po linii wpisujac je do zmiennej LINIA
while read -r LINIA; do
     # Tak zapisujemy dodanie dwoch zmiennych. Zliczamy tutaj sume od 1 do 10
     SUMA=$((SUMA + LINIA))
done <<< "$ZMIENNA"
# Wypisujemy sume na ekran
echo "Suma to: $SUMA"
```

Ćwiczenie: billing telefoniczny

W ramach tego ćwiczenia spróbujemy połączyć nabytą przez nas wiedzę, aby stworzyć coś użytecznego. Na początek, pobieramy plik `billing.txt` z materiałów do zajęć i zapisujemy go w katalogu `~/billing/` (prawym → zapisz element docelowy jako...).
Plik `billing.txt` zawiera szczegóły tysiąca rozmów telefonicznych. Na nieszczęście, ktoś bardzo nieumiejętnie sformatował ten plik i dane tam zawarte są zupełnie nieczytelne. Na szczęście zaś - umiemy bash'a i poradzimy sobie z tym.

1) Zapoznaj się z zawartością pliku. Jakie dane się tam znajdują? Jak oddzielone są kolejne wpisy?
2) Stwórz pusty skrypt o nazwie `skrypt.sh` w katalogu `~/billing/`. Nadaj mu odpowiednie uprawnienia i otwórz go w edytorze tekstu Notepad++. Tryb pracy jest następujący: układamy sobie na jednym ekranie trzy okienka, aby nie trzeba było się ciągle przełączać: konspekt, Notepad++ i konsolę. Wprowadzamy zmianę do skryptu, zapisujemy, uruchamiamy skrypt w konsoli i oceniamy wynik działania. Na razie nasz skrypt powinien mieć nagłówek i jedną linijkę, którą będziemy rozbudowywać o poniższe funkcjonalności:
3) Cały billing jest zestawem wpisów rozdzielonych odpowiednim znakiem i wpisanych do jednej linii. Spowoduj, aby każdy wpis zaczynał się w nowej linii.
4) W naszym billingu coś jest nie tak z odstępami pomiędzy kolejnymi słowami. Napraw to tak, aby zawsze między słowami był odstęp dokładnie jednej spacji.
5) Teraz powinniśmy mieć skrypt, który wypisuje na ekran ładnie sformatowaną tabelkę z wpisami billingowymi. Rozszerz go o jeszcze jedną komendę, która spowoduje, że wypisywane będzie tylko pierwsze 20 linijek. Dzięki temu łatwiej będzie ocenić efekt kolejnych modyfikacji.
6) Od tego momentu, nasz skrypt będzie musiał się rozrastać do wielo-linijkowego, aby możliwe było zrobienie następnych zadań.
7) Posortuj tabelkę billingową. Ale uwaga, zwykłe posortowanie spowoduje, że linijka z nagłówkiem przeskoczy nam na koniec. Zrób tak, aby wyjście ze skryptu zawierało linijkę nagłówkową zawsze na początku, a dalej posortowane wpisy.
8) Obecnie numery telefonów są w postaci `xxx-xxx-xxx`. Zmień ich postać na `xxxxxxxxx`. Uważaj, aby nie zmienić nagłówka tabelki!
9) W nazwiskach abonentów wdarło się kilka pomyłek – napraw je:
   1) Grzegosz Tomczak → Grzegorz Tomczak
   2) Marcin Workowski → Marcin Borkowski
   3) Marzena Pieprzak → Marzena Pietrzak
   4) Krystyna Palczak → Krystyna Walczak
10) Dodaj do skryptu opcję --osoba, która wypisze tylko połączenia w których brała udział dana osoba. Wywołanie powinno wyglądać następująco:

```
~$ ./skrypt.sh --osoba "Krystyna Walczak"
```

11) Usuń komendę ograniczającą wpisy do 20 linijek. Rozszerz podpunkt 10) w taki sposób, aby pod listą połączeń pojawiło się podsumowanie (oddzielone nową linią). Powinno to działać w następujący sposób:

```
~$ ./skrypt.sh --osoba "Krystyna Walczak"
...
23:35:28 630394400 Krystyna Walczak 641321224 Beata Wlodarczyk 281
23:59:43 637159662 Jakub Sokolowski 630394400 Krystyna Walczak 232

Abonent:     Krystyna Walczak
Czas rozmow: 32752 sekund
```

12) Rozszerz podpunkt 11) o wnioskowanie numeru telefonu abonenta:

```
~$ ./skrypt.sh --osoba "Krystyna Walczak"
...
23:35:28 630394400 Krystyna Walczak 641321224 Beata Wlodarczyk 281
23:59:43 637159662 Jakub Sokolowski 630394400 Krystyna Walczak 232

Abonent:        Krystyna Walczak
Numer telefonu: 630394400
Czas rozmow:    32752 sekund
```

13) Rozszerz podpunkt 12) o ładne formatowanie czasu rozmów (podpowiedź: posłuż się komendą **date**).

```
~$ ./skrypt.sh --osoba "Krystyna Walczak"
...
23:35:28 630394400 Krystyna Walczak 641321224 Beata Wlodarczyk 281
23:59:43 637159662 Jakub Sokolowski 630394400 Krystyna Walczak 232

Abonent:        Krystyna Walczak
Numer telefonu: 630394400
Czas rozmow:    09:05:52
```

14) Rozszerz podpunkt 13) o zliczanie ilości rozmów:

```
~$ ./skrypt.sh --osoba "Krystyna Walczak"
...
23:35:28 630394400 Krystyna Walczak 641321224 Beata Wlodarczyk 281
23:59:43 637159662 Jakub Sokolowski 630394400 Krystyna Walczak 232

Abonent:        Krystyna Walczak
Numer telefonu: 630394400
Ilosc rozmow:   69
Czas rozmow:    09:05:52
```

15) Dodaj opcję `--wychodzace`, która wypisze tylko rozmowy wychodzące od zadanej osoby i wyświetli podsumowanie.

```
~$ ./skrypt.sh --wychodzace "Krystyna Walczak"
...
21:52:54 630394400 Krystyna Walczak 632038265 Jozef Nowakowski 974
23:35:28 630394400 Krystyna Walczak 641321224 Beata Wlodarczyk 281

Abonent:                   Krystyna Walczak
Numer telefonu:            630394400
Liczba rozmow wychodzacych: 28
Czas rozmow wychodzacych:  03:35:07
```

16) Dodaj opcję `--odebrane`, analogicznie do poprzedniego podpunktu.

```
~$ ./skrypt.sh --odebrane "Krystyna Walczak"
...
23:07:12 640842382 Grzegorz Tomczak 630394400 Krystyna Walczak 326
23:59:43 637159662 Jakub Sokolowski 630394400 Krystyna Walczak 232

Abonent:                   Krystyna Walczak
Numer telefonu:            630394400
Liczba rozmow odebranych:  41
Czas rozmow odebranych:    05:30:45
```

17) Rozszerz opcję --osoba o podsumowanie rozmów wychodzących i odebranych:

```
~$ ./skrypt.sh --osoba "Krystyna Walczak"
...
23:35:28 630394400 Krystyna Walczak 641321224 Beata Wlodarczyk 281
23:59:43 637159662 Jakub Sokolowski 630394400 Krystyna Walczak 232

Abonent:                    Krystyna Walczak
Numer telefonu:             630394400
Liczba rozmow odebranych:   41
Czas rozmow odebranych:     05:30:45
Liczba rozmow wychodzacych: 28
Czas rozmow wychodzacych:   03:35:07
Liczba rozmow w sumie:      69
Czas rozmow w sumie:        09:05:52
```

Ćwiczenie 2: billing w pythonie

Kolejnym zadaniem jest stworzenie analogicznego skryptu w języku python. W tym celu stwórz nowy skrypt w pythonie, tworząc pusty plik pod ścieżką: `~/billing/skrypt.py`. Otwórz go do edycji w edytorze typu Notepad++. Skrypt można uruchomić komendą:

```
~$ python skrypt.py
```

Wykonaj zadania z poprzedniego ćwiczenia, aby uzyskać takie samo wyjście ze skryptu dla poszczególnych zadań. Poniżej lista funkcji pythonowych, które mogą się okazać przydatne:

```
# Wypisuje zadany napis na konsole
print('napis')

# Formatuje napis wstawiajac w miejsce klamr wartosci zmiennych
'to jest formatowany {} zawierajacy liczbe {}'.format('napis', 17)

# Pozwala poznac ilosc argumentow programu (sys.argv jest tablica)
import sys
...
liczba_arg = len(sys.argv)

# Formatuje czas HH:MM:SS przeliczając sekundy
import datetime
...
str(datetime.timedelta(seconds=liczba_sekund))

# Wczytuje plik do zmiennej tekstowej >content<
with open('plik.txt', 'r') as file_to_read:
     content = file_to_read.read()

# Dzieli tekst na przecinkach i wynikowi umieszcza w zmiennej tablicowej
tablica = tekst.split(',')

# Sortuje tablice
tablica.sort()

# Zamienia wystapienia slowa 'wzorzec' na 'podstawienie' w zmiennej tekstowej
nowy_tekst = tekst.replace('wzorzec', 'podstawienie')

# Laczy tablice napisow w jeden napis za pomoca przecinkow
nowy_tekst = ','.join(['napis1','napis2','napis3'])
```

billing z gwiazdką*

W wybranym języku (bash / python) – a najlepiej obu - napisz obsługę opcji `--podsumowanie`, która podliczy billing dla wszystkich osób:

```
~$ ./skrypt.sh --podsumowanie
...
23:35:28 630394400 Krystyna Walczak 641321224 Beata Wlodarczyk 281
23:59:43 637159662 Jakub Sokolowski 630394400 Krystyna Walczak 232

Abonent          | Nr tel     | Wychodzace     | Odebrane       | Sumarycznie
-----------------|------------|----------------|----------------|---------------
Joanna Krajewska | 662792392  | 10:31:24 (60)  | 05:36:13 (44)  | 16:07:37 (104)
Elzbieta Sawicka | 601253419  | 05:58:22 (47)  | 05:02:37 (42)  | 11:00:59 (89)
Katarzyna Kubiak | 690794243  | 10:53:10 (57)  | 10:01:12 (41)  | 20:54:22 (98)
Celina Grabowska | 664852169  | 07:42:19 (54)  | 08:56:23 (62)  | 16:38:42 (116)
Barbara Krawczyk | 649447179  | 06:04:36 (44)  | 06:18:31 (54)  | 12:23:07 (98)
Bogdan Kolodziej | 637550141  | 10:27:44 (55)  | 09:25:45 (64)  | 19:53:29 (119)
Jakub Sokolowski | 637159662  | 06:58:11 (50)  | 07:36:04 (55)  | 14:34:15 (105)
Agnieszka Wojcik | 697446556  | 07:35:28 (59)  | 07:05:34 (40)  | 14:41:02 (99)
Grzegorz Tomczak | 640842382  | 08:07:52 (62)  | 07:17:33 (53)  | 15:25:25 (115)
Marzena Pietrzak | 651148812  | 07:30:03 (55)  | 06:48:53 (54)  | 14:18:56 (109)
Bartosz Domagala | 667624781  | 10:00:06 (48)  | 06:30:25 (42)  | 16:30:31 (90)
Marzena Szewczyk | 682821675  | 05:00:47 (41)  | 05:14:42 (57)  | 10:15:29 (98)
Marcin Borkowski | 623680830  | 09:29:12 (61)  | 07:14:43 (46)  | 16:43:55 (107)
Tomasz Michalski | 631894734  | 05:30:49 (45)  | 06:24:58 (52)  | 11:55:47 (97)
Michal Czarnecki | 686824068  | 07:38:10 (60)  | 07:44:37 (47)  | 15:22:47 (107)
Jozef Nowakowski | 632038265  | 05:34:59 (45)  | 06:56:22 (54)  | 12:31:21 (99)
Krzysztof Wrobel | 607441476  | 05:13:52 (46)  | 08:03:46 (58)  | 13:17:38 (104)
Beata Wlodarczyk | 641321224  | 03:48:34 (44)  | 03:35:38 (43)  | 07:24:12 (87)
Krystyna Walczak | 630394400  | 03:35:07 (28)  | 05:30:45 (41)  | 09:05:52 (69)
Mateusz Dziedzic | 633327378  | 05:05:06 (39)  | 11:21:10 (51)  | 16:26:16 (90)
```

Polecam wysłać sobie napisany skrypt na maila!

Wszystko to już umiem, nie mam co robić

Poniżej wrzucam parę tutoriali, które można sobie przejrzeć i/lub przerobić:

1. Tutorial ogólny: http://ryanstutorials.net/linuxtutorial/
2. Kurs online: https://www.codecademy.com/courses/learn-the-command-line/
3. Przypomnienie regular expressions:
    1. http://www.regular-expressions.info/tutorial.html
    2. https://regexone.com/
    3. https://regexcrossword.com/
4. GREP
    1. http://opensourceforu.com/2012/06/beginners-guide-gnu-grep-basics-regular-expressions/
    2. http://evc-cit.info/cit052/grep1.html (wykonać tylko polecenia dotyczące grep).

Zadanie domowe dla chętnych. Zainstaluj sobie linuxa! Najłatwiej zainstalowac program VirtualBox na swoim Windowsie i wybrać jakąś dystrubucję linuxa z listy dostępnych obrazów systemowych (może to wymagać pobrania obrazu), uruchomić i gotowe. Na początek proponuję dystrybucję **Ubuntu**. Zapoznać się z systemem, oswoić się z poznanymi komendami.

https://www.virtualbox.org/