THEORETICAL COMPUTER SCIENCE
Jagiellonian University

polecenia powłoki 2

`less` to program do przeglądania plików tekstowych w konsoli. **Uwaga:** `less` nie jest edytorem. Jedną z jego zalet jest to, że nie wczytuje on całego pliku do pamięci przed wyświetleniem, dzięki czemu szybciej działa od standardowych edytorów na dużych plikach. Standardowe wywołanie programu to:

```
less filename
```

Oto lista niektórych komend przy nawigacji:

↑ lub ↓	przesunięcie o jedną linię,
`/wzorzec`	wyszukanie wzorca od bieżącej linii do przodu,
`n`	wyszukanie następnego wystąpienia wzorca,
`N`	wyszukanie poprzedniego wystąpienia wzorca,
`F`	przejście do trybu oczekiwania na dopisywanie linii do pliku,
`q`	wyjście z programu.

A oto kilka użytecznych wywołań `less`:

```
less +150 filename
less +G filename
less +F filename
```

`tr` to program do zamiany lub usuwania znaków. Składnia jego wywołania jest następująca: `tr [opcje] "set1" "set2"`, gdzie `set1` jest zbiorem znaków, które chcemy zmienić w odpowiadające im znaki w zbiorze `set2`. Jeżeli w użyciu jest opcja `-d` (usuwanie znaków), to `"set2"` nie występuje. Przykładowe wywołania:

```
echo "abczdefcba azaaz" | tr "abc" "def"
⟶defzdeffed dzddz
echo "abczdefcba azaaz" | tr "abcz" "def"
⟶deffdeffed dfddf
echo "abczdefcba azaaz" | tr -d "abc"
⟶zdef zz
```

Zamiast wypisywać kolejne znaki, możemy zastosować zbiory:

```
echo 'Linux' | tr "a-z" "A-Z"
echo 'Linux' | tr "[:lower:]" "[:upper:]"
```

Opcja `-c` sprawia, że polecenie `tr` zamiast zbioru `set1` przyjmuje jego dopełnienie. Opcja `-s` powoduje zamianę **ciągu** znaków ze zbioru `set1`, na **jeden** odpowiadający mu znak ze zbioru `set2`.

```
tr -cs "[:alpha:]" "\n" < input.txt
tr -cd "[:print:]" < input.txt
```

`diff` to program do porównywania zawartości dwóch plików. Wypisuje listę modyfikacji (`a` – dodanie linii, `c` – zamiana linii, `d` – usunięcie linii), które trzeba zastosować do linii w pliku pierwszym, aby uzyskać plik drugi. Standardowe wywołanie:

```
diff plik1 plik2
```

Oto lista niektórych przydatnych opcji dla polecenia `diff`:

`-b`	ignoruje zmiany w liczbie białych znaków,
`-B`	ignoruje puste linie,
`-i`	ignoruje wielkość liter,
`-I regexp`	ignoruje linie pasujące do wyrażenia regularnego `regexp`,
`-q`	podaje tylko informację czy pliki się różnią,
`-r`	rekurencyjnie porównuje zawartość dwóch katalogów,
`-u`	zmienia format raportu porównania na zwarty kontekstowy,
`-w`	ignoruje białe znaki.

`iconv` to program zmieniający kodowanie znaków danego pliku na inne. Przykładowo, zmiana z `ISO-8859-1` na `UTF-8` wygląda następująco:

```
iconv -f iso-8859-1 -t utf-8 <input.txt >output.txt
```

Listę dostępnych kodowań można zobaczyć po wywołaniu:

```
iconv --list
```

`aspell` to program do sprawdzania pisowni, a przede wszystkim wychwytywania literówek. Program ma filtry dopasowane do plików `html`, `tex/latex` i kilku innych. Do sprawdzania pisowni można podłączyć zewnętrzne słowniki. Poprawianie błędów jest wykonywane poprzez interaktywną formułę: `aspell` wskazuje błąd oraz podobne słowa w słowniku, użytkownik decyduje czy (i co) robić ze wskazanym słowem.

Przykłady użycia:

```
aspell check file.txt
aspell check --lang=pl file.txt
```

Poprzez komendę `pipe` program `aspell` wskazuje literówki bez interakcji, co jest bardziej wskazane, jeśli chcemy dokonać analizy statystycznej naszych błędów. Zatem polecenie `aspell pipe < file.txt` wydrukuje na standardowe wyjście jedną linię dla każdego słowa w pliku `file.txt`. Jeśli słowo jest w słowniku linia będzie zawierać tylko jeden znak `*`. Jeśli słowo nie jest w słowniku, ale `aspell` ma sugestie co to powinno być, wtedy linia wygląda tak:

```
& original count offset: miss, miss, ...
```

Jeśli słowo nie jest w słowniku i `aspell` nie ma sugestii, wtedy linia wygląda tak:

```
# original offset
```

`ssh` to program umożliwiający połączenie z serwerem przy pomocy protokołu *secure shell*. Protokół ten jest następcą protokołu *Telnet* i służy do terminalowego łączenia się ze zdalnym komputerami. Obecnie *SSH* to wspólna nazwa dla całej większej rodziny protokołów (*SCP*, *SFTP*). Wspólną cechą tych protokołów jest technika szyfrowania i rozpoznawania użytkownika.

Podstawowe użycie programu `ssh` wygląda tak:

```
ssh username@student.tcs.uj.edu.pl
```

Przy pierwszej próbie połączenia z danym serwerem `ssh` poda jak się przedstawia znaleziony zdalny komputer i czy na pewno chcemy się połączyć oraz dołożyć jego i jego klucz RSA do listy znanych hostów (`~/.ssh/known_hosts`).
`ssh` umożliwia również uwierzytelnianie poprzez klucz publiczny i prywatny. Jest to bez-

piecznejszy i często wygodniejszy sposób w praktyce. Aby wygenerować klucze piszemy: `ssh-keygen -t dsa` ... i wpisujemy tzw. *passphrase*. Z założenia powinna to być dłuższa sekwencja znaków od standardowego hasła. Najczęściej jest to zdanie w języku naturalnym. Klucz prywatny i publiczny domyślnie zapisują się w katalogu `~/.ssh/` w plikach `id_dsa` i `id_dsa.pub`, odpowiednio. Następnie powinniśmy skopiować klucz publiczny do odpowiedniego katalogu na zdalnym komputerze:

```
scp ~/.ssh/id_dsa.pub username@student.tcs.uj.edu.pl:.ssh/authorized_keys
```

Jedną z mniej znanych zalet *X windows* jest ich niesamowita transparentność w sieci. Ten system okienkowy z założenia miał być gotowy transmisji *GUI* i bitmap. W skrócie jeśli mamy na lokalnym i zdalnym komputerze *X windows* to możemy połączyć się przez `ssh` i odpalić zdalnie np. przeglądarkę internetową. Połączenie tunelujące protokół *X11* włączamy opcją `-X`:

```
ssh -X username@student.tcs.uj.edu.pl
```

Można też nawiązywać połączenie ze zdalnym komputerem celem wykonania jednego polecenia w terminalu. Jest to szczególnie użyteczne, gdy jesteśmy na maszynie A, chcemy pracować na maszynie C i jedyny sposób połączenia się z tą maszyna wiedzie poprzez maszynę B.

```
ssh username@student.tcs.uj.edu.pl ls -al
ssh username@B ssh username@C
```

`scp` to program, który używa protokołu *SSH* do przesyłania plików pomiędzy lokalnym i zdalnym komputerem.

```
scp filename username@student.tcs.uj.edu.pl:
scp filename username@student.tcs.uj.edu.pl:directory/new_filename
scp username@student.tcs.uj.edu.pl:filename newfilename
scp -r directory username@student.tcs.uj.edu.pl:
```

`wget` to prosty program do pobierania plików za pośrednictwem protokołu *HTTP* lub *FTP*.

```
wget http://srodowisko.tcs.uj.edu.pl/docs/srodowisko-2012-grep.pdf
wget -O taglist.zip http://.../download_script.php?id=1234
wget -c http://...
wget -b http://...
tail -f wget-log
wget -r http://...
```

bash, aliasy i zmienne

Przy pomocy polecenia `alias` można definiować nowe polecenia na bazie już istniejących. Samo polecenie `alias` bez argumentów wyświetla wszystkie komendy powłoki, które przy jego pomocy powstały. Przykładowo, jeżeli często używamy polecenia `ls -al`, to możemy je zastąpić jednym poleceniem `lsa` w następujący sposób:

```
alias lsa='ls -al'
```

Od tej pory, aż do zamknięcia powłoki, będziemy mogli używać samego polecenia `lsa` do dokładnego wypisywania zawartości aktualnego katalogu czy `lsa bin` do podobnego wyświetlania katalogu `bin`. W celu utrzymania tej komendy na stałe (czyli po zamknięciu i ponownym uruchomieniu basha) powinniśmy jej powyższą definicję wpisać do osobnej linijki w pliku `~/.bashrc`. W nim znajdują się zdefiniowane przez użytkownika funkcje, aliasy i zmienne, które nabierają znaczenia przy starcie basha. A zrestartować powłokę możemy przy pomocy polecenia:

```
exec bash
```

Wspomniane *zmienne* możemy traktować jak etykiety, które przechowują ciągi znaków. Definiujemy je przy pomocy znaku **= bez spacji po obu jego stronach**:

```
dog="Azor"
```

Do zmiennych w poleceniach odwołujemy się poprzedzając je symbolem `$`. Okazuje się, że to, czy nazwę zmiennej umieścimy pomiędzy apostrofami, czy też cudzysłowami, ma znaczenie – cudzysłów umożliwia specjalną interpretację znaku `$`.

```
echo $dog
⟶Azor
echo $dogek
⟶
echo ${dog}ek
⟶Azorek
echo 'Jestem $LOGNAME, a to jest pies $dog.'
⟶Jestem $LOGNAME, a to jest pies $dog.
echo "Jestem $LOGNAME, a to jest pies $dog."
⟶Jestem kosinski, a to jest pies Azor.
```

Powyżej pojawiła się, zawierająca nazwę użytkownika, zmienna `LOGNAME` – jedna ze *zmiennych środowiskowych*. Są to zmienne, których zawartość jest dostępna dla wszystkich procesów uruchomionych w powłoce. Przyjęło się, że ich nazwy zapisujemy dużymi literami. Przykłady innych zmiennych środowiskowych:

`HOSTNAME`	nazwa (adres) maszyny, na którą się zalogowaliśmy,
`HOME`	ścieżka do katalogu domowego użytkownika,
`PATH`	ścieżki (oddzielone dwukropkami), w których szukane są programy powłoki,
`PWD`	ścieżka do aktualnego katalogu.

Wszystkie zmienne środowiskowe możemy wypisać komendą `env` (choć nie jest to jej główne przeznaczenie – patrz `man env`). W praktyce opłaca się nam często zmodyfikować zmienną `PATH` w pliku `~/.bashrc` w celu szybkiego uruchamiania naszych programów (skryptów) z danego katalogu:

```
PATH=$PATH:$HOME/scripts
```

Jeżeli w katalogu `scripts` znajdował się program o nazwie `cmd`, to od tej pory będziemy go mogli uruchomić w dowolnym miejscu wpisując po prostu `cmd`.

Polecenie `history` wyświetla historię wydanych poleceń w bashu. Może ona zawierać bardzo dużo wpisów, zatem najlepiej wywoływać ją np. w postaci `history | less` lub `history n`, gdzie `n` jest liczbą ostatnich poleceń (czyli `history 30` wypisze 30 ostatnio używanych komend). Do zawartości historii możemy odnosić się następująco:

`!!` ostatnio używane polecenie,
`!5` piąte polecenie z historii,
`!-3` trzecie od końca polecenie,
`!l` ostatnie polecenie zaczynające się na "l",
`!*` lista argumentów (poza zerowym) ostatniego polecenia,
`!l:1` pierwszy argument ostatniego polecenia zaczynającego się na "l",
`!!:1-3` argumenty od 1 do 3 ostatniego polecenia.

Przez "argumenty" należy rozumieć kolejne wyrazy. Na przykład w poleceniu:
`cat -nE plik`
argumentem 0 jest `cat`, argumentem 1 – `-nE`, a argumentem 2 – `plik`. Warto zauważyć, że polecenia są zapamiętywane w historii już z podstawionymi konkretnymi instrukcjami (argumentami) w miejscu tych zaczynających się od `!`.

Historia wywołanych komend basha znajduje się domyślnie w pliku `~/bash_history`. Zawartość tego pliku może być jednak mniej aktualna od informacji z polecenia `history`, gdyż aktualizowana jest standardowo przy zamykaniu powłoki. Aby natychmiast zaktualizować plik `~/bash_history` należy użyć komendy: `history -a`

procesy

Proces to instancja wykonywanego programu. Systemy operacyjne mogą obsługiwać wiele procesów naraz, każdemu z nich przydzielając numer identyfikacyjny – *PID* (ang. *process identifier*) oraz odpowiednie zasoby, takie jak pamięć czy czas procesora.

Poniżej podajemy przykładowy proces, który powinien trwać na tyle długo, aby miało sens mierzenie mu czasu komendą `time` – próbuje przeglądać zawartość katalogu głównego i rekurencyjnie zawartość jego podkatalogów, ale nie będzie nic wypisywał na ekran konsoli (symbol `&>` oznacza jednoczesne przekierowanie standardowego wyjścia danych `>` i standardowego wyjścia błędów `2>` do podanego pliku – w tym wypadku specjalnego pliku, który ignoruje wszystkie dane zapisane do niego):

```
time ls -R / &> /dev/null
⟶
⟶real    0m2.594s
⟶user    0m1.234s
⟶sys     0m1.336s
```

Czas `real` to faktyczny czas wykonania programu, czas `user` to czas poświęcony na wykonywaniu samych instrukcji programu bez ingerencji w polecenia systemu operacyjnego, natomiast czas `sys`, to czas poświęcony na wykonywanie instrukcji w jądrze systemu operacyjnego.

Trochę zmodyfikujemy nasze ostatnie polecenie, aby zapisywało swój rezultat do normalnego pliku: `ls -R / &> ~/katalogi`
Z poziomu konsoli możemy przerwać działanie tego programu przy pomocy kombinacji Ctrl-C. Po wciśnięciu klawiszy wyemitowany zostanie sygnał `SIGINT`, który powinien spowodować wymienione przerwanie, o ile nie zostanie wcześniej przechwycony. Za inną kombinacją, Ctrl-Z, kryje się sygnał `SIGTSTP`, który wstrzymuje działanie programu. Program wstrzymany można ponownie uruchomić od momentu, w którym nastąpiło wstrzymanie.

Jeśli użyliśmy tej ostatniej kombinacji, to zobaczymy status zadania – zatrzymane.

Przy pomocy polecenia `jobs` możemy zobaczyć stan wszystkich zadań w aktualnej sesji. Możemy uruchomić i zatrzymać w podany sposób kolejny proces: `sleep 1000` ("nic nie rób przez 1000 sekund"), co pozwoli nam zobaczyć dwa numerowane wpisy na liście zadań (pytanie: jaki będzie rezultat wykonania `time sleep 5`?). Jeśli chcemy zakończyć pierwszy z tych procesów, to możemy zrobić to przy pomocy polecenia `kill %1` odwołującego się do numeru na liście zadań poprzedzonego znakiem `%`.

Alternatywnie, jako argument polecenia `kill` możemy użyć numeru PID. Uzyskamy go dzięki poleceniu `ps`, które wypisuje listę naszych procesów związanych z konsolą. Inne przykłady wywołań polecenia `ps`:

`ps x`	wszystkie nasze procesy, także niezwiązane z konsolą;
`ps ux`	jak wyżej, wraz dodatkowymi informacjami, np. o właścicielu procesu i czasie startu;
`ps ax`	wszystkie procesy wszystkich osób na serwerze;
`ps axo pid,user,tty,command`	to co wyżej, ale w czterech następujących kolumnach: PID, nazwa użytkownika, konsola, treść komendy.

Teraz możemy np. usunąć proces o numerze PID 4493 poprzez komendę `kill 4493`. Polecenie `kill` standardowo wysyła sygnał `SIGTERM`, który jednak, podobnie jak `SIGINT`, może zostać przechwycony przez program. W przypadkach krytycznych należy zatem używać opcji `-9` (np. `kill -9 4493`), powodującej wysłanie sygnału `SIGKILL`, który działa jak `SIGTERM`, ale nie da się go przechwycić. Jeżeli naszym celem jest zakończenie od razu wszystkich procesów związanych z konkretną komendą, to możemy użyć polecenia `killall`, np. `killall sleep`.

```
#include<stdio.h>
#include<signal.h>

void sighandler(int sig) {
  printf("Caught signal %d. Avoiding ^C.\n", sig);
}

int main() {
  signal(SIGINT, &sighandler);
  while(1);
  return 0;
}
```

Istniejące zadanie (np. o numerze 2 na liście `jobs`) można zmusić do pracy w tle przy pomocy polecenia `bg %2`. Jeśli jednak od razu chcemy wykonać dane zadanie w tle, to wystarczy na jego końcu wstawić znak `&`. **Uwaga:** procesy działające w tle będą działały nadal po zakończeniu sesji! Przykład polecenia:

```
ls -l / &> ~/katalogi2 &
⟶[3] 5988
```

Od razu otrzymaliśmy w ten sposób numer zadania i numer PID. Przyjmijmy, że po wywołaniu komendy `jobs` otrzymamy:

```
⟶[1]+  Stopped                 ls -R / >&katalogi
⟶[2]   Running                 sleep 1000 &
⟶[3]-  Done                    ls -l / >&katalogi2
```

Zadanie o numerze 3 zakończyło się, więc przy kolejnym wywołaniu `jobs` już nie pojawi się na liście. Drugie zadanie działa w tle, natomiast pierwsze wciąż jest zatrzymane – możemy je przenieść do pracy na pierwszym planie przy pomocy polecenia `fg %1`.

Zarządzać procesami można przy użyciu poleceń `top` lub `htop`. Aby uzyskać pomoc w trakcie działania któregoś z tych programów, wciskamy jeden z klawiszy `h` lub `?`. Wyjście odbywa się przy pomocy `q`. `htop` jest bardziej przejrzysty i wygodny w obsłudze niż `top`, ale ten drugi z kolei oferuje możliwość zapisu raportu do pliku. Poniższe akapity są poświęcone programowi `top`.

W celu zobaczenia procesów należących wyłącznie do użytkownika `user1` możemy użyć komendy: `top -u user1`. Alternatywnie, w trakcie działania programu `top` wciskamy klawisz `u`, wpisujemy `user1` i akceptujemy. Jeżeli nie napiszemy żadnej nazwy, to zobaczymy procesy wszystkich użytkowników. Kończymy działanie danego procesu wciskając `k` i podając jego PID. Przy pomocy polecenia `top -p pid1, pid2, ...` (np. `top -p 1, 3`) możemy śledzić wyłącznie procesy o podanych PIDach (w przykładzie: `1` i `3`). Klawisze `z` i `b` wpływają na oznaczenie aktywnych procesów.

Procesy możemy sortować względem wybranej kolumny. Domyślnie posortowane są po procentowym zużyciu procesora (klawisz `P`). Jeśli chcemy ułożyć je po zużyciu pamięci, to używamy klawisza `M`. W przypadku innego sortowania używamy `O` lub `F`, natomiast `R` odwraca sortowanie. Możemy także decydować o wyborze kolumn do wyświetlenia (`f`) i ich kolejności (`o`). Klawisz `c` (jak i opcja `-c` przy uruchomieniu) sprawia, że możemy przełączać się między mniej i bardziej szczegółowym widokiem komend związanych z poszczególnymi procesami. Przy pomocy `l`, `t` i `m` można włączać lub wyłączać podsumowania nad listą procesów.

Domyślny czas, po którym odświeżana jest lista procesów wynosi 3 sekundy. Aby go zmodyfikować, używamy albo komendy `top -d liczba_sekund`, albo klawiszy `d` lub `s` w trakcie działania polecenia `top`. Jeżeli chcemy natychmiast poznać aktualny stan listy procesów, to używamy enter bądź spacji.

W celu zapisania aktualnego stanu procesów do pliku używamy polecenia:

```
top -bn 1 > plik
```

Wykonywać różne zadania w tym samym czasie z przełączaniem się na pierwszy plan i tło można także przy użyciu polecenia `screen`. W praktyce ten program tworzy w osobnych procesach wirtualne terminale, między którymi można się łatwo przemieszczać. Podobnie jak w przypadku procesów działających w tle, wirtualne terminale będą istnieć także po zakończeniu sesji.

Taki wirtualny terminal można zainicjalizować, wpisując `screen komenda` lub po prostu `screen`, a potem wpisując komendę. W tym pierwszym przypadku jednak wirtualny

terminal zniknie wraz z zakończenie procesu. Przykładowo:

```
screen sleep 2000
```

Aby wrócić do głównego terminala, należy użyć kombinacji `Ctrl-a d`. W ten sposób można opuścić wirtualny terminal, jednocześnie nie usuwając go. Przekonamy się o tym, wpisując komendę `screen -ls`, która wylistuje aktualne sesje `screen`a. Wcześniej jednak wywołamy `screen -S nazwa`, która utworzy nam wirtualny terminal o ustalonej nazwie i opuścimy go. Zatem:

```
screen -ls
⟶There are screens on:
⟶        5506.pts-0.tcs  (Detached)
⟶        8367.nazwa      (Detached)
⟶2 Sockets in /tmp/uscreens/S-kosinski.
```

Poszczególne `screeny` wypisane są w postaci: `PID.nazwa_terminala[.host]`. Jeżeli chcemy wejść do jednego z tych wirtualnych terminali (powiedzmy: `5506.pts-0.tcs`), to wpisujemy jedno z następujących poleceń (ostatnie dwa z nich zadziałają, o ile nazwa wraz z ewentualnym hostem jest unikalna):

```
screen -r 5506.pts-0.tcs
screen -r 5506
screen -r pts-0
screen -r pts-0.tcs
```

Ciekawostka: możemy opuścić aktualny wirtualny terminal z innego... "fizycznego" terminala, wpisując `screen -d identyfikator`, gdzie `identyfikator` jest taki, jak w przypadku opcji `-r`.

Wewnątrz `screen`a tworzymy kolejne wirtualne podterminale przy pomocy kombinacji `Ctrl-a c`. Możemy się między nimi przemieszczać przy pomocy kombinacji:

`Ctrl-a n`	następny podterminal,
`Ctrl-a p`	poprzedni podterminal,
`Ctrl-a 0-9`	skok do podterminala o podanym numerze (od 0 do 9),
`Ctrl-a '`	skok do podterminala o numerze podanym po `'`,
`Ctrl-a "`	przejście do menu wyboru podterminala,
`Ctrl-a Ctrl-a`	przemieszczenie się między 2 ostatnio używanymi podterminalami.

Ogólnie wewnątrz `screen`a używamy poleceń, które zaczynają się od `Ctrl-a` (patrz pomoc – `Ctrl-a ?`). W celu zamknięcia podterminala używamy `Ctrl-a K`, ewentualnie komendy `exit`. Po zamknięciu wszystkich podterminali w danym `screen`ie, ten ostatni kończy działanie.

Zadanie 1. Napisz prosty program w `C++` dopisujący co pewien czas nowe linie do pliku `przyklad.txt`. Uruchom ten program w tle. Użyj programu `less` do obserwacji zmian w pliku `przyklad.txt`

Zadanie 2. Podobnie jak w poprzednim zadaniu napisz i uruchom dwa programy w tle dopisujące nowe linie do dwóch ustalonych plików. Użyj programu `tail`, aby obserwować na bieżąco zmiany w obu tych plikach jednocześnie.

Zadanie 3. Wypisz liczbę literówek znalezionych przez `aspell` w kopii książki Conrada.

Zadanie 4. Wypisz 10 najczęstszych literówek w kopii książki Conrada. Wyjście ma składać się z 10 linii. W pojedynczej linii ma być liczba powtórzeń danej literówki oraz samo potencjalnie błędne słowo.

Zadanie 5. Połącz się poleceniem `ssh` z dowolnym zdalnym komputerem. Zamknij połączenie i przejrzyj zawartość pliku `~/.ssh/known_hosts`. Użyj programu `sed` do przefiltrowania pliku `~/.ssh/known_hosts` i wydrukowania dla każdego znanego hosta dwóch linii: w pierwszej linii ma się znaleźć nazwa hosta, w drugiej jego adres IP.

Zadanie 6. Użyj programu `ssh` do uruchomienia przeglądarki internetowej na zdalnym komputerze.

Zadanie 7. Stwórz nietrywialne drzewo katalogów na zdalnym komputerze. Przekopiuj rekurencyjnie katalog główny Twojego drzewa na maszynę lokalną, używając programu `scp`.

Zadanie 8. Napisz alias dla programu `sed` z opcją `-r` i uczyń go permanentnym, tzn. zapisz go tak, aby móc go odpalić również z innej powłoki.

Zadanie 9. Napisz alias dorzucający bieżący katalog do listy katalogów, w których powłoka szuka programów do wykonania.

Zadanie 10. Wydrukuj ranking 10 najczęściej używanych programów/poleceń na podstawie historii powłoki. Wyjście ma zawierać 10 linii. W pojedynczej linii ma być liczba powtórzeń wykonania programu oraz nazwa programu.

Zadanie 11. Wydrukuj ranking 10 najczęściej używanych programów/poleceń na podstawie historii powłoki. Tym razem uwzględnij również programy wywołane po użyciu potoków, czyli znaku `|`.

Zadanie 12. Wydrukuj ranking 10 najczęściej używanych programów/poleceń na podstawie historii powłoki. W tym rankingu uwzględnij tylko te programy, które użyte są po użyciu potoku (czyli wszystkie poza pierwszym we wprowadzonej linii).

Zadanie 13. Napisz program w `C/C++` przechwytujący sygnały `SIGINT` i `SIGTERM`. Zademonstruj jego działanie. Później dopisz do programu przechwytywanie sygnału `SIGKILL`. Zademonstruj ...

Zadanie 14. Ile procesów ma uruchomionych na serwerze studenckim użytkownik `root`?

Zadanie 15. Ilu różnych użytkowników ma uruchomione procesy na serwerze studenckim?

Zadanie 16. Ile niekonsolowych procesów (symbol `?` w kolumnie `TTY`) jest łącznie uruchomionych na serwerze studenckim?

Zadanie 17. Ile konsolowych procesów mają łącznie uruchomionych na serwerze studenckim użytkownicy o loginach składających się z siedmiu cyfr poprzedzonych literą "z"?

Zadanie 18. Wypisz listę 10 w danej chwili najczęściej używanych komend (wraz z argumentami) w procesach na serwerze studenckim wraz z ich częstotliwością.

Zadanie 19. Wypisz listę 5 w danej chwili najczęściej używanych komend (pomijając argumenty, czyli wszystko po pierwszej spacji) o PIDach zaczynających się od "3" lub "4" w procesach na serwerze studenckim wraz z ich częstotliwością.