Wprowadzenie do systemu Unix
Podstawowe komendy cz. 2.

Wprowadzenie

W niniejszej instrukcji zajmiemy się wydobywaniem określonych informacji z plików lub poleceń systemowych. Często zawartość takiego pliku lub wyjście odpowiedniego polecenia zawiera dużo informacji, które w danym momencie nie są nam potrzebne. W celu uzyskania interesującej nas zawartości, konieczne jest filtrowanie tekstu, ograniczające rozmiar wyjścia.

Zazwyczaj konieczne okazuje się przetwarzanie ciągu znaków (po prostu tekstu) wieloma poleceniami, połączonymi symbolem *pipe* (|), pozwalającym na tzw. *przetwarzanie potokowe*. Wynikiem takiego postępowania jest gotowa linia komend. Gdy pojawia się pytanie o linię komend, to jako odpowiedź musisz podać jedną, linię polecenia. Nie możesz używać żadnych znaków Enter – poza ostatnim, zatwierdzającym.

Podstawowym narzędziem służącym do wyszukiwania wzorców jest **grep**. Pozwala on na pozyskiwanie danych ze standardowego wejścia (np. poprzez przekierowanie) lub podanej ścieżki. Polecenie wykorzystuje wyrażenia regularne, czyli język opisu wzorców wyszukiwania.

Dużo przydatnych informacji o wyrażeniach regularnych i poleceniu **grep** można znaleźć na stronie: `http://tldp.org/LDP/Bash-Beginners-Guide/html/chap_04.html`.

W wyrażeniu regularnym większość znaków odpowiada po prostu sobie. Ciąg znaków **abc** będzie dopasowywał po prostu takie znaki następujące po sobie. Istnieje jednak wiele znaków specjalnych, na przykład:

1. ^ – początek linii.
2. **$** – koniec linii.
3. **.** – jeden dowolny znak.
4. **[ ]** – oznacza grupę znaków i dopasowuje jeden z podanych znaków. Dla przykładu: **[abc]** dopasowuje jeden z znaków „a”, „b” lub „c”. Można też definiować zakresy znaków: **[a-f]** oznacza jeden ze znaków z przedziału a-f, a **[1-3]** – dowolną cyfrę z przedziału 1-3.
5. **[^]** – dopasowuje jeden znak spoza podanej grupy znaków. Na przykład **[^0-9]** dopasuje dowolny znak nie będący cyfrą.
6. **|** – to alternatywa logiczna. Dopasowuje ciąg po lewej albo po prawej. Na przykład, **ab|cd** dopasuje ciąg **ab** albo **cd**.
7. **( )** – grupuje podwyrażenia. Jest to przydatne m.in. jeżeli chcemy wymusić określoną kolejność interpretacji wyrażenia. Na przykład **a(b|c)** spowoduje dopasowanie zarówno ciągu **ab** jak i **ac**. Bez grupowania (**ab|c**) miałoby interpretację **ab** albo **c**.

Backslash (\) powoduje anulowanie interpretacji następnego znaku jako specjalnego. Przykładowo, ciąg znaków **\$** zostanie zinterpretowany dosłownie jako znak dolara ($), a nie znak końca linii.

W wyrażeniu regularnym można także określać liczbę wystąpień danego znaku lub zgrupowanego nawiasami wzorca:

1. **a?** – brak lub jedno wystąpienie,
2. **a*** – brak lub dowolna liczba wystąpień,
3. **a+** – co najmniej jedno wystąpienie,
4. **a{3}** – dokładnie 3 wystąpienia,
5. **a{5,}** – co najmniej 5 wystąpień,
6. **a{3,5}** – między 3 a 5 (włącznie) wystąpień.

Należy pamiętać, że niektóre znaki specjalne występujące w wyrażeniach regularnych traktowane są jak zwykłe znaki, jeśli nie jest podana opcja **-E**, która uruchamia obsługę wyrażeń rozszerzonych. Na zajęciach rekomendujemy stosowanie **-E** cały czas.

INNYM NARZĘDZIEM, lepiej dopasowanym do przetwarzania bardziej ustrukturyzowanych plików tekstowych, jest **awk**. Jest to wszechstronny język programowania służący do przetwarzania danych tekstowych i dopasowywania wyrażeń regularnych.

Nazwa AWK pochodzi od pierwszych liter nazwisk autorów: Alfreda V. Aho, Petera Weinbergera i Briana Kernighana.

Składnia AWK jest bardzo prosta. Składa się z par warunków i list akcji:

```
warunek { akcja; akcja; }
warunek { akcja; akcja; }
```

Warunki dopasowują najczęściej fragment tekstu.

1. Zapoznaj się z manualem dla komendy **awk** i spróbuj odpowiedzieć na następujące pytania:
   - W jaki sposób tworzone są skrypty AWK?
   - Jakie wykorzystywać zmienne określające linie i kolumny?
   - Jak wypisać konkretną kolumnę?
   - Jak określić znacznik podziału kolumn?
   - Jak konstruuje się instrukcje warunkowe i dopasowania wzorców?
   - Jak dopasować początek i koniec przetwarzania?
2. Spróbuj napisać własny skrypt **awk** i wypisać tylko identyfikatory użytkowników (UID) z pliku `/etc/passwd`.

ABY DOPASOWYWAĆ NAZWY I ŚCIEŻKI PLIKÓW stosujemy *globbing*. Wykorzystując wbudowane opcje powłoki systemu operacyjnego, pozwala on na interpretację argumentów danej komendy i rozwinięcie jej poprzez dopasowanie odpowiednich nazw plików. Pozwala to na wykonanie polecenia na większym zbiorze argumentów (plików) bez konieczności korzystania z bardziej zaawansowanych metod, np. komendy **find**.

Szczegółowy opis metody *pathname expansion* można znaleźć pod adresem: `http://wiki.bash-hackers.org/syntax/expansion/globs`.

We wzorcach rozpoznawane są:

1. * – dowolna ilość dowolnych znaków (w tym ich brak),
2. **?** – dokładnie jeden dowolny znak,
3. **[abc]** – dokładnie jeden z podanych znaków,
4. **[a-z]** – dokładnie jeden ze znaków mieszczących się w podanym zakresie,
5. **[!abc]** – dokładnie jeden znak inny niż podane,
6. **[!a-z]** – dokładnie jeden ze znaków niemieszczących się w podanym zakresie.

NA POPRZEDNICH ZAJĘCIACH przedstawione zostało już jedno przekierowanie – *pipe*, pozwalające na przekazanie wyjścia polecenia po lewej stronie znaku | na wejście komendy po jego prawej stronie:

```
wyjście_stąd | wchodzi_tutaj
```

Oprócz *potoku*, istnieją także inne przekierowania, jak pokazuje Tabela 1.

Symbol	Znaczenie
<	Przekierowanie zawartości pliku na wejście polecenia
>	Przekierowanie wyjścia polecenia do pliku (z nadpisaniem dotychczasowej zawartości tego pliku)
>>	Doklejenie wyjścia polecenia na koniec wskazanego pliku

Tabela 1: Przekierowania.

Przekierowania można łączyć, zatem poprawne jest następujące wywołanie:

```
polecenie < wejście >> wyjście_doklejane
```

Można posługiwać się także krótkimi oznaczeniami strumieni (Tabela 2).

Strumień	Wartość	Znaczenie
STDIN	0	Standardowy strumień wejścia
STDOUT	1	Standardowy strumień wyjścia
STDERR	2	Standardowy strumień wyjścia błędów (diagnostycznego)

Tabela 2: Podstawowe strumienie w programie.

Przedstawione poniżej polecenie można zatem odczytać jako procedurę rekursywnego kasowania plików z katalogu `/tmp`, do których wykonujący komendę użytkownik ma prawo zapisu („w"), a błędy wywołania zostaną wysłane do pliku `err.log` w katalogu domowym użytkownika:

```
rm -rf /tmp 2> $HOME/err.log
```

Przekierowania mogą być także złączane. Służy do tego znak *ampersand* (`&`). Zatem, polecenie analogiczne do poprzedniego, ale pozwalające wysłać komunikaty o błędach do wskazanego pliku i jednocześnie wysłać tam też normalne wyjście, przedstawia się następująco:

```
rm -rf /tmp 2> $HOME/err.log 1&>2
```

Zadania

Operacje na tekście:

Część spośród widocznych tutaj poleceń pojawiła się już w poprzedniej instrukcji jako zadania dodatkowe.

1. Zapoznaj się z poleceniami **tr** oraz **cut**. W jaki sposób mogą one zastąpić podstawowe funkcje dostarczane przez **awk**?
2. Jak w systemie opisany został (zawartość piątej kolumny pliku `/etc/passwd`) użytkownik, którego numer identyfikacyjny (UID) wynosi 14?

3. Ilu użytkowników zdefiniowanych w systemie używa jako podstawowego interpretera poleceń interpretera Bash?
4. Ilu użytkowników w systemie posiada numer identyfikacyjny (UID) większy od 20?
5. Jaki jest największy numer grupy lokalnej w systemie?
6. Ile jest w systemie grup, które nie są grupą podstawową dla ani jednego jednego użytkownika?
7. Ile grup w systemie nie jest grupami dodatkowymi dla żadnego użytkownika?
8. Ilu jest użytkowników w systemie, których grupa podstawowa jest Twoją grupą?
9. Wypisz użytkowników, których nazwa składa się z dokładnie 8 znaków.
10. W katalogu domowym utwórz plik zawierający strony manuala dla komendy **ls**. Jakie są różnice w działaniu operatorów przekierowania > oraz >>?

Operacje na plikach:

1. Zapoznaj się z poleceniami **mkdir**, **rmdir** oraz **ls**. Następnie, wykorzystując jedno wywołanie polecenia **mkdir**, utwórz w swoim katalogu osobistym strukturę katalogów tak, aby poprawne były następujące ścieżki:
   - `$HOME/c2`
   - `$HOME/c2/text`
   - `$HOME/c2/bin`
2. Sprawdź prawa dostępu katalogów. Co decyduje o ich postaci? Czy stojąc w katalogu `$HOME` jednym wywołaniem komendy **ls** można obejrzeć zawartość wszystkich katalogów do samego końca drzewa plików? Sprawdź to w *manualu*.
3. Sprawdź prawa dostępu utworzonego w poprzednim zadaniu pliku. Zapoznaj się z poleceniami **chmod** i **chown**. Zmień prawa dostępu pliku tak, aby tylko jego właściciel indywidualny mógł go odczytać i zapisać.
4. Ustaw prawa dostępu pliku i katalogu, w którym plik się znajduje, tak, aby pliku nie można było usunąć komendą **rm**. Skopiuj plik na dowolny inny. Jakie są prawa dostępu nowego pliku? Zmień prawa dostępu nowego pliku tak, aby tylko właściciel indywidualny i inni członkowie jego grupy podstawowej mogli go odczytać.
5. Skopiuj z katalogu `/bin` do swojego katalogu `~/c2/bin` plik `ls`, zmieniając jego nazwę na `moj_ls`. W jaki sposób można skopiować cały katalog `/usr/include/scsi`?

Zadanie sprawdzające

Poniższe zadanie wykorzystuje kompleksowo wiedzę z tego laboratorium. Postaraj się wykonać je samodzielnie w domu. Jeżeli masz z nim problemy, przestudiuj ponownie materiały źródłowe, rozwiąż wcześniejsze zadania i podejmij kolejną próbę.

Podaj pełną komendę zwracającą konkretną wartość (nie należy np. liczyć wierszy „ręcznie"):

1. Ile jest takich grup, które są grupami podstawowymi dla dwóch lub trzech użytkowników?
2. Podaj liczbę lokalnych użytkowników, którzy nie mogą się zalogować (`/sbin/nologin` jako shell)?
3. Ile jest osób w `/etc/passwd` o nazwisku Nowak?
4. Ilu użytkowników było jednorazowo zalogowanych co najmniej godzinę (wykorzystaj wyniki polecenia **last**)?

Podsumowanie komend

Na zajęciach przedstawione zostały następujące komendy:

Grupa	Komenda
Przetwarzanie tekstu	awk, uniq, sort, tr, cut
Wyszukiwanie w tekście	grep
Uprawnienia	chmod, chown
Wyszukiwanie plików	find
Operacje w systemie plików	cp, mkdir, rmdir, touch, rm

Istotna poznana składnia: **<**, **>**, **>>**, **2>**, **2>>**, **2&>1**.

Przydatne informacje: składnia Basha i znaki specjalne

Zwróć uwagę na różnice pomiędzy pojedynczymi a podwójnymi cudzysłowami. Jak traktowane są znaki specjalne znajdujące się w ich obrębie? Dużo informacji znajdziesz w podręczniku użytkownika dla powłoki *Bash*: `https://www.gnu.org/software/bash/manual/html_node/Shell-Syntax.html`

Wykonywanie operacji matematycznych z poziomu powłoki jest możliwe dzięki mechanizmowi *arithmetic expansion*. Pozwala ono na wykonanie operacji i zwrócenie jej wyniku. Poniżej przedstawiono proste wyrażenie zwracające wynik dodawania dwóch liczb:

O mechanizmie *arithmetic expansion* oraz innych pokrewnych można przeczytać tutaj `http://wiki.bash-hackers.org/syntax/expansion/intro`.

```
echo $(( 5 + 2 ))
```

Argumenty operacji mogą być także rezultatem innych poleceń, tworząc bardziej skomplikowaną strukturę (tutaj pozwalającą wyznaczyć sumę liczb wierszy plików `/etc/passwd` oraz `/etc/group`):

```
echo $(( $(cat /etc/passwd | wc -l) + $(cat /etc/group | wc -l) ))
```

Oczywiście, nie jest to najprostsza metoda realizacji tego zadania, taką samą informację można pozyskać w poniższy sposób:

```
cat /etc/passwd /etc/group | wc -l
```

Opisana jest jednak dużo bardziej ogólna i znacznie częściej można ją zastosować.

PRZYDATNYM MECHANIZMEM jest również tzw. *command substitition*. Pozwala ono na wykorzystanie w komendzie wyjścia zwróconego przez inną komendę, wykonaną wcześniej. Istnieją dwie składnie:

1. Bez możliwości zagnieżdżania: `komenda_zewnętrzna` `komenda_wewnętrzna`
   Trochę wydumany przykład (zmiana właściciela pliku na aktualnego użytkownika): `sudo chown` `id -u` `file`
2. Z możliwością zagnieżdżania: `komenda_zewnętrzna $(komenda_wewnętrzna)`
   Przykład: `sudo chown $(id -u) file`