UnixLinux phần 3

Xử lý dữ liệu qua Command line

Đinh Nhật Thành

Ngày 2 tháng 7 năm 2025

Mục lục

Giải thích thuật ngữ cơ bản

Thuật ngữ	Giải thích
Unix/Linux	Là các hệ điều hành mã nguồn mở, được xem là nền tảng cho nhiều công cụ phân tích dữ liệu hiện đại nhờ sự ổn định và hiệu quả cao.
Shell	Một chương trình giao diện dòng lệnh, đóng vai trò trung gian giúp người dùng giao tiếp và ra lệnh cho hệ điều hành. Các loại phổ biến bao gồm `Bash`, `Zsh`, và `Fish`.
Terminal	Là cửa sổ hoặc ứng dụng nơi người dùng gõ các lệnh để `Shell` thực thi.
Distributions (Bản phân phối)	Là các phiên bản khác nhau của hệ điều hành Linux, được xây dựng và tùy chỉnh cho các mục đích sử dụng đa dạng. Ví dụ: `Ubuntu`, `Debian`, `CentOS`, `WSL`.
API (Application Programming Interface)	có thể hiểu là "Giao diện cho BackEnd"vì nó cho phép ta gọi 1 hàm để 'Thêm hoặc Xem hoặc Sửa hoặc Xóa' từ 1 database hoặc file dữ liệu từ bất cứ đâu thông qua 1 đường link. Giả dụ bạn muốn code sử dụng cả Python lẫn Java, bạn có thể gọi và sử dụng output của 1 hàm Python trong Java như bình thường.
CSV (Comma-Separated Values)	Định dạng tệp văn bản để lưu trữ dữ liệu dạng bảng, trong đó các giá trị được phân tách bằng dấu phẩy, rất phổ biến trong lưu trữ dữ liệu.
JSON (JavaScript Object Notation)	Định dạng văn bản dùng để trao đổi dữ liệu có cấu trúc giống dictionary trong Python với mỗi giá trị đều có 1 cặp Key và Value, thường được các API sử dụng để trả về kết quả.
Lệnh cơ bản	`cd`, `pwd`, `ls`, `mkdir`: `cd`: lệnh cơ bản để điều hướng, `pwd`: hiển thị vị trí của hiện tại, `ls`: liệt kê và tạo thư mục. `touch`, `cp`, `mv`, `rm`: Các lệnh để tạo, sao chép, di chuyển (đổi tên), và xóa tệp/thư mục.
Lệnh xem và tìm kiếm dữ liệu cơ bản	`cat`, `head`, `tail`, `less`: Các lệnh dùng để xem nội dung tệp. `cat` hiển thị toàn bộ, `head`/`tail` xem phần đầu/cuối, và `less` xem theo từng trang. `find`, `grep`: Các lệnh để tìm kiếm tệp theo tiêu chí và tìm các dòng chứa một mẫu (pattern) nhất định.
Lệnh xử lý dữ liệu cơ bản	`wc`, `sort`, `uniq`: `wc` là lệnh để đếm (dòng, từ, ký tự), `sort` sắp xếp dữ liệu và `uniq` để lọc/đếm các dòng trùng lặp. `cut`, `sed`, `awk`: `cut` Công cụ mạnh mẽ để trích xuất dữ liệu theo cột, `sed` chỉnh sửa văn bản theo luồng và `awk` là lệnh nâng cao để xử lý dữ liệu có cấu trúc như bảng chứa trong file .csv hoặc database.
Lệnh quản lý hệ thống thông dụng	`ps`, `top`, `kill`: Các lệnh để quản lý các tiến trình (process) đang chạy, `ps` giúp theo dõi, `top` kiểm soát và `kill` dừng các tác vụ. `chmod` là change mode dùng để thay đổi quyền truy cập , `chown` là change owner dùng để thay đổi quyền sở hữu của tệp/thư mục.
`vi/vim`	Trình soạn thảo văn bản (i.e. IDE) hoạt động ngay trong terminal, giống như vscode nhưng mọi tính năng đều sử dụng thông qua terminal

Thuật ngữ	Giải thích
Biến môi trường (Environment Variable)	Các biến chứa thông tin về môi trường làm việc của shell, dùng để khai báo đường dẫn và hoạt động của các chương trình phân tích dữ liệu để dùng sau này. e.g. Biến `PATH` nói với Shell đường tìm lệnh `python`
Pipe (|)	Ký hiệu dùng để kết nối output của một lệnh với input của lệnh khác, tạo thành một chuỗi xử lý (pipeline) (giống như dây truyền sản xuất logistic).
Redirect (>, >>, <)	Các ký hiệu dùng để chuyển hướng đầu vào/đầu ra của một lệnh tới tệp thay vì màn hình.
`Tham số (Parameter)`	là 1 cái tên đại diện cho giá trị đầu vào của 1 hàm, có thể coi nó là 1 variable được khai báo nhưng không chứa giá trị e.g. `def ham(tham_so)`
`Đối số (Argument)`	là giá trị được truyền vào tham số khi 1 hàm được gọi e.g. `ham(doi_so)`
`stdin`, `stdout`, `stderr`	Ba luồng dữ liệu chuẩn trong Linux: đầu vào chuẩn, đầu ra chuẩn, và đầu ra lỗi chuẩn.
`xargs`	Lệnh nhận đầu vào và thực thi một lệnh khác với các đầu vào đó làm đối số i.e. arg .
`jq`	Công cụ chuyên dụng để phân tích, lọc và biến đổi dữ liệu từ các tệp JSON. e.g. `jq .name data.json` lấy giá trị "name"từ tệp JSON
`wget`, `curl`	công cụ để tải dữ liệu từ Internet, với `wget` phù hợp cho tệp lớn và `curl` linh hoạt cho API. e.g. `wget http://example.com/file.txt.` và `curl http://api.example.com`
`tar`, `gzip`, `zcat`	Lệnh để đóng gói (`tar`), nén (`gzip`), và xem nội dung tệp nén (`zcat`) mà không cần giải nén hoàn toàn.

Mở rộng phần 2: User Permission

Cấu trúc quyền trong Linux		Lệnh phân quyền	
rwx	read (đọc dữ liệu), write (ghi/sửa dữ liệu), execute (chạy scripts)	**chmod 755**	Cho phép chạy script Python xử lý dữ liệu
u	user (người sở hữu)	**chmod u+x**	Thêm quyền thực thi cho script huấn luyện model
g	group (nhóm)	**chown**	Chuyển quyền sở hữu dataset
o	others (người dùng khác)	**chmod 400**	Bảo vệ file chứa API keys chỉ để đọc
rw-r--r--	File dữ liệu chỉ có thể đọc với group và others		

Hình 1: Quyền và phân quyền files đối cho từng loại người dùng user-group-others

Tóm gọn hình trên, ta thấy:
3 đối tượng người dùng là `u, g, o` đại diện cho `user, group, other` có vị trí mặc định là `chmod ugo` trong chmod. Tương tự, 3 lệnh phân quyền là `r, w, e` đại diện cho `read, write, execute` cũng có vị trí mặc định là `rwx`. Nhưng nếu không muốn sử dụng cả 3 lệnh phân quyền, mình có thể dùng dấu – để thể hiện

cho quyền rỗng, ví dụ `r-x` nghĩa là đối tượng có quyền read và execute nhưng không thể write.

- `rw=6` (read write), textttr-x=5 (read, no write permission, execite), textttr–=4 (only read, no write and execute), texttt—=0 (no persmission).
- Có thể đặt bằng số như 1 biến textttx=1, w=2, r=5, rwx=7 (read write execute). Với ugo là vị trí của các đối tượng trong chmod, chmod 742 ghĩa là user có quyền read write execute và group cùng others đều có quyền read.

Hướng tiếp cận các câu lệnh hiệu quả: các câu lệnh đều là các keyword viết tắt, mình có thể dễ dàng hiểu nó qua việc dịch. Ví dụ: `chmod` nghĩa là change mode, `chowner` nghĩa là change owner.

Phần 3: Xử lý dữ liệu qua Command Line

Note: giải thích lý do sử dụng -> syntax -> giải thích ngữ nghĩa sâu sau.

1 Tại sao lại Xử lý dữ liệu qua Command Line (Ưu thế và bất lợi)

- **Ưu thế**: Dữ liệu được xử lý nhanh chóng vì command line loại bỏ giao diện đồ họa, giúp tiết kiệm tài nguyên và tăng tốc độ thao tác. Các lệnh có thể kết hợp linh hoạt (e.g. sử dụng chaining/piping) để xử lý dữ liệu phức tạp chỉ với một dòng lệnh. Ngoài ra, command line dễ tự động hóa bằng script, phù hợp cho xử lý dữ liệu lớn hoặc lặp đi lặp lại.
- **Bất lợi**: Command line có thể khó tiếp cận với người mới do cú pháp phức tạp, dễ mắc lỗi khi thao tác nhất là khi mới làm quen với Regex Express. Bạn sẽ cần phải tự viết 1 chương trình để trực quan hóa rồi chạy nó trong command line (e.g. ảnh, video) . Ngoài ra, bạn sẽ phải thực hành hằng ngày để ghi nhớ các lệnh tốt.
- **VSCode vs Command Line**: VSCode cung cấp giao diện đồ họa thân thiện, hỗ trợ nhiều tính năng chạy ngầm như gợi ý mã, kiểm tra lỗi, hay CoPilot và tích hợp git. Nhưng command line loại bỏ hoàn toàn những tính năng đó và chỉ tập trung vào việc xử lý chương trình. Ví dụ: VSCode có thể ngốn từ 1-3GB RAM khi chạy code vì các tính năng background trong khi Command Line chỉ cần vài trăm MB

2 Regex Expression

Regex (Regulat Expression - Biểu thức chính quy) là 1 cách **mô tả văn bản** theo một cấu trúc văn bản (text pattern) nhất định, được tích hợp trong Linux, Regex cho phép mình **tìm kiếm, thay thế hoặc phân tích các cấu trúc dữ liệu 1 cách linh hoạt** .Nếu bạn theo lập trình lâu dài, regex chắn chắn là công cụ không thể thiếu.

Ví dụ, với regex bạn có thể tìm hoặc thay thế mọi từ có các kí hiệu 'xyz' nối sau như "helloxyz", "muaxyz", "lolxyz"hoặc thậm chí là trích xuất các hàng trong bảng theo 1 cấu trúc khi kết hợp với lệnh Linux, miễn là trích xuất thông tin có xử lý text, regex sẽ cho bạn làm điều đó hay bất kì cấu trúc nào khác phức tạp hơn 1 cách dễ dàng.

Để tiếp cận regex 1 cách hiệu quả và vui nhất, bạn có thể bắt đầu bằng cách áp dụng các regex syntax đơn giản để truy cứu, thay thế những cấu trúc câu hoặc thực hành thêm ở RegexOne

Cách sử dụng Regex trong Command Line: giống như 1 ngôn ngữ lập trình, regex có thể được dùng ngay trong command line, hoặc trong 1 file.

Ví dụ: Sử dụng Regex với grep

```
#!/bin/sh
# In regex, "." represents any single character. For example, "App.e" will match "Apple", "Appie", "Appae", etc., as long as it starts with "App" and ends with "e".

# Read the text file using cat
# In Linux, start the command with grep to use Regex Expression
fruits_file=`cat fruit.txt | grep App.e`

# Here, the base word (result) will be Apple,
# but since we don't know the exact spelling of Apple,
# we put a dot (.) in that position.

echo "Use '.' to find the base word, when the given word is 'App.e'"

# Display the result
echo "Result:"

echo "$fruits_file"
```

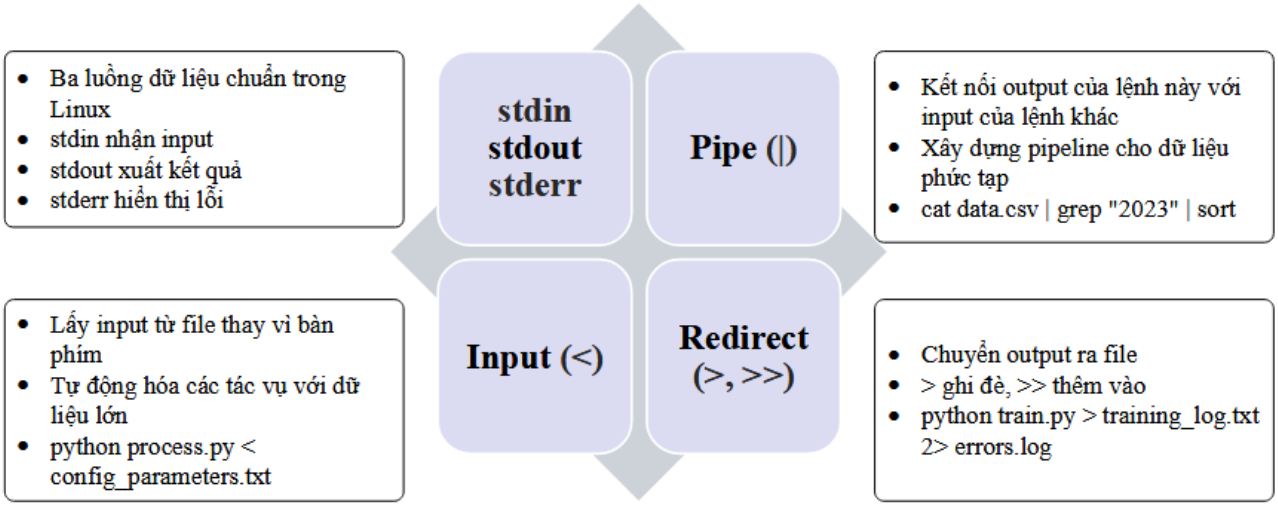
- **Ký tự đại diện (Wildcard Characters):**
  - `.` (dấu chấm): Khớp với bất kỳ ký tự nào (trừ ký tự xuống dòng). Ví dụ: `a.b` khớp với ”axb”, ”ayb”, ”a.b”.
  - `*` (dấu hoa thị): Khớp với 0 hoặc nhiều lần xuất hiện của ký tự hoặc nhóm đứng trước nó. Ví dụ: `a*` khớp với ””, ”a”, ”aa”, ”aaa”. `.*` khtớp với bất kỳ chuỗi ký tự nào.
  - `+` (dấu cộng): Khớp với 1 hoặc nhiều lần xuất hiện của ký tự hoặc nhóm đứng trước nó. Ví dụ: `a+` khớp với ”a”, ”aa”, ”aaa”.
  - `?` (dấu hỏi): Khớp với 0 hoặc 1 lần xuất hiện của ký tự hoặc nhóm đứng trước nó (tức là tùy chọn). Ví dụ: `colour` khớp với ”color”và ”colour”.
  - `[}^]*\` (bất kỳ kí tự nào): đại diện cho mọi Pattern, khi mình muốn truy vấn pattern `function(...)` mà không quan tâm đến bên trong có gì, ta có thể dùng `[}^]*\` để đại diện cho mọi pattern hay kí tự bên trong `function(...)`
- **Neo (Anchors):**
  - `^` (dấu mũ): Khớp với sự bắt đầu của một dòng. Ví dụ: `^Home` khớp với ”Home”chỉ khi nó ở đầu dòng.
  - `$` (dấu đô la): Khớp với sự kết thúc của một dòng. Ví dụ: `End$` khớp với ”End”chỉ khi nó ở cuối dòng.
  - `\b`: Khớp với ranh giới từ (word boundary). Ví dụ: `\bcat\b` chỉ khớp với từ ”cat”, không phải ”category”.
- **Lớp ký tự (Character Classes):**
  - `[abc]`: Khớp với bất kỳ ký tự nào trong dấu ngoặc vuông. Ví dụ: `[aeiou]` khớp với bất kỳ nguyên âm nào.
  - `[a-z]`: Khớp với bất kỳ ký tự nào trong phạm vi được chỉ định. Ví dụ: `[0-9]` khớp với bất kỳ chữ số nào.

- `[^abc]` : Khớp với bất kỳ ký tự nào KHÔNG nằm trong dấu ngoặc vuông. Ví dụ: `[^0-9]` khớp với bất kỳ ký tự nào không phải là chữ số.
- `\d` : Khớp với bất kỳ chữ số nào (tương đương với `[0-9]`).
- `\D` : Khớp với bất kỳ ký tự nào không phải chữ số (tương đương với `[^0-9]`).
- `\w` : Khớp với bất kỳ ký tự chữ-số hoặc dấu gạch dưới (tương đương với `[a-zA-Z0-9_]`).
- `\W` : Khớp với bất kỳ ký tự nào không phải chữ-số hoặc dấu gạch dưới.
- `\s` : Khớp với bất kỳ ký tự khoảng trắng nào (space, tab, newline, v.v.).
- `\S` : Khớp với bất kỳ ký tự nào không phải khoảng trắng.

- **Nhóm (Groups) và Tham chiếu ngược (Backreferences):**
  - `(pattern)` : Tạo một nhóm chụp (capturing group) cho một phần của biểu thức. Ví dụ: `(abc)+` khớp với ”abc”, ”abcabc”.
  - `\n` : Tham chiếu ngược đến nội dung đã được nhóm chụp thứ `n`. Ví dụ: `(\w+)\s+\1` khớp với các từ lặp lại như ”hello hello”.
- **Lượng tử (Quantifiers):**
  - `{n}` : Khớp chính xác `n` lần. Ví dụ: `a{3}` khớp với ”aaa”.
  - `{n,}` : Khớp ít nhất `n` lần. Ví dụ: `a{2,}` khớp với ”aa”, ”aaa”, v.v.
  - `{n,m}` : Khớp ít nhất `n` lần và nhiều nhất `m` lần. Ví dụ: `a{1,3}` khớp với ”a”, ”aa”, ”aaa”.
- **Ký tự thoát (Escaping Characters):**
  - `\` : Ký tự gạch chéo ngược dùng để thoát các ký tự có ý nghĩa đặc biệt trong Regex, biến chúng thành ký tự nghĩa đen. Ví dụ: `\.` khớp với dấu chấm nghĩa đen, không phải bất kỳ ký tự nào.

3 Pipe và Redirect

Trong 4 khối lệnh, **Pipe** (`|`) và **Redirect** (`>`, `>>`, `<`) là hai công cụ rất quan trọng trong Unix/Linux, cho phép bạn nối các lệnh lại với nhau và điều khiển luồng dữ liệu đầu vào/đầu ra.



Hình 2: 4 khối lệnh cơ bản để xây dựng pipeline xử lý dữ liệu

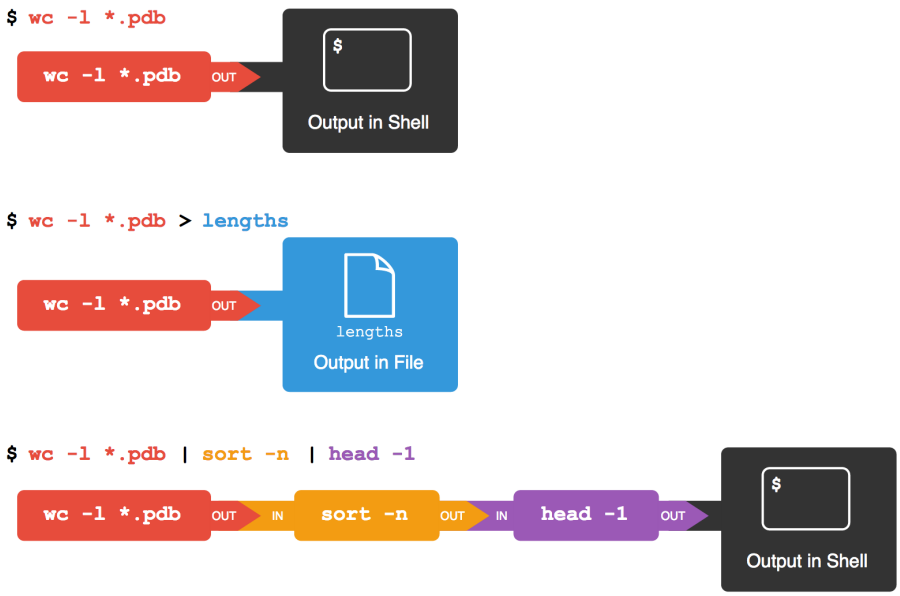
3.1 Minh họa flow hoạt động của Pipe và Redirect

Lệnh Pipe lấy trong từ pipeline, trong thực tế 1 đường ống lớn là sự kết hợp của các đường ống nhỏ gồm có đầu vào và đầu ra nối lại với nhau. Tương tự như thế, Pipe có thể hiểu là các đường ống (pipelines) hay 1 hàm gồm input và output được nối lại với nhau tạo thành 1 pipe hoàn chỉnh.

Trong hình minh họa, ta có hàm `wc -l *.pdb` dùng để đếm số dòng trong mỗi file, trong đó:

- `wc` nghĩa là word count.
- `-l` nghĩa là list dùng đẻ liệt kê.
- `*.pdb` có * nghĩa là tất cả, `*.pdb` nghĩa là cho mọi file có đuôi `.pbd`.

Nếu không làm gì, output của hàm sẽ được tự động viết ra Shell, giống như trong ngôn ngữ C, mình có thể dùng mũi tên để điều chỉnh hướng của dữ liệu là ghi đè (append) output vào file (note: < để lấy input, > để ghi đè, » để thêm vào).



Hình 3: Minh họa về Pipe và Redirect trong Command Line

Cuối cùng, để kết hợp nhiều lệnh với nhau thành 1 pipe, mình dùng | để nối đầu ra của lệnh trước `wc -l *.pdb` tới đầu vào của lệnh sau là `sort -n` trong đó `-n` là 1 flag thông báo sort theo thứ tự số nhỏ tới lớn thay vì chữ từ a-z. Vậy "`wc -l *.pdb | sort -n`"nghĩa là đếm mọi số dòng trong mọi file có đuôi `.pdb` rồi sort theo dòng từ nhỏ tới lớn. Cuối cùng mình sử dụng `head -1` để lấy file có số dòng nhỏ nhất.

3.2 Ví dụ các bước xây dựng flow hoạt động của 1 Pipeline đơn giản

Ví dụ 1: Pipe đơn giản – lọc dữ liệu Giả sử bạn có file `users.csv` chứa danh sách người dùng, bạn muốn lọc ra các dòng chứa từ "admin".

```
cat users.csv | grep "admin"
```

Giải thích:

- `cat users.csv`: đọc toàn bộ nội dung file.
- `grep "admin"`: lọc các dòng có chứa từ ”admin”.
- `|`: kết nối hai lệnh – nội dung từ `cat` được chuyển cho `grep`.

Ví dụ 2: Redirect – ghi kết quả vào file

Bây giờ bạn muốn lưu kết quả lọc trên vào file `admins.txt`:

```
cat users.csv | grep "admin" > admins.txt
```

Giải thích:

- > redirect output sang file mới tên là `admins.txt`.
- Nếu file đã tồn tại, nó sẽ bị ghi đè.

Ví dụ 3: Kết hợp pipe và append

Lưu tiếp kết quả các dòng chứa “moderator” vào cùng file:

```
cat users.csv | grep "moderator" >> admins.txt
```

Giải thích:

- >> thêm nội dung vào cuối file `admins.txt`, không ghi đè.

Tóm tắt

Pipe và redirect giúp bạn kết nối các công cụ nhỏ để xử lý dữ liệu một cách linh hoạt, ví dụ như:

```
cat data.csv | grep "2024" | cut -d',' -f1 | sort | uniq > results.txt
```

Câu lệnh trên trích xuất cột 1 từ các dòng chứa ”2024”, sắp xếp và lọc trùng, sau đó ghi vào file `results.txt`.

4 Filter Commands cho Data – Bộ lọc mạnh mẽ của Command Line

Command Line trong Unix/Linux cung cấp các công cụ xử lý văn bản cực kỳ mạnh mẽ như `sed`, `awk`, `cut`, `tr`, và `grep`. Đây là các lệnh thường dùng để lọc, chỉnh sửa và trích xuất thông tin từ dữ liệu dạng văn bản (logs, CSV, JSON, v.v.). Mình có thể hình dung cách sử dụng cơ bản của từng lệnh với các ví dụ đơn giản sau:

1. `grep` – Tìm dòng chứa mẫu (pattern)

```
grep "ERROR" logfile.txt
```

Tìm tất cả các dòng trong file `logfile.txt` có chứa từ `ERROR`.

2. `cut` – Cắt cột hoặc ký tự trong dòng

```
cut -d',' -f1,3 data.csv
```

Trích xuất cột 1 và 3 (phân tách bằng dấu phẩy) từ file CSV.

3. `awk` – Xử lý theo dòng và cột như một mini-script

```
awk '{print $2}' users.txt
```

In cột thứ hai trong mỗi dòng của file `users.txt` (dấu cách là mặc định).

4. `sed` – Tìm và thay thế văn bản

```
sed 's/admin/user/g' accounts.txt
```

Thay mọi từ ”admin”thành ”user”trong file `accounts.txt` và in ra kết quả.

5. `tr` – Chuyển đổi ký tự

```
tr 'a-z' 'A-Z' < names.txt
```

Chuyển toàn bộ chữ thường thành chữ in hoa trong file `names.txt`.

Mở rộng: Các công cụ trên có thể kết hợp với nhau qua `pipe (|)` để xây dựng quy trình xử lý dữ liệu phức tạp. Hãy thử kết hợp chúng để:

- Lọc các dòng có năm 2023 trong file logs.
- Trích cột thứ ba và chuyển thành chữ in hoa.
- Tự động thay thế định dạng ngày từ DD/MM/YYYY sang YYYY-MM-DD.

5 Trích xuất dữ liệu từ files lớn

6 Xử lý dữ liệu JSON với jq

6.1 Xử lý dữ liệu JSON với jq phần 1

6.2 Xử lý dữ liệu JSON với jq phần 2

7 Xử lý CSV qua command line

8 Phân tích logs với command line

9 Kết hợp các commands phức tạp

10 xargs - xử lý dữ liệu song song

11 Xử lý dữ liệu thực tế

12 Tải dữ liệu từ internet (wget, curl)

13 Làm việc với files né

14 giải nén và xử lý nhiều files

15 Bài tập thực hành

1. **Xử lý file CSV người dùng**
   Xử lý `users_2023-01-01.csv`: Đếm người dùng theo khu vực. Lọc người dùng theo trạng thái "Active". Sắp xếp theo tần suất truy cập (giảm dần).

2. **Phân tích dữ liệu JSON thời tiết**
   Phân tích file `weather_hanoi.json`. Trích xuất nhiệt độ, độ ẩm theo ngày. Tính nhiệt độ và độ ẩm trung bình.

3. **Tạo báo cáo hiệu suất mô hình**
   Phân tích file `model_logs.txt` để tổng hợp precision, recall theo ngày và phát hiện xu hướng suy giảm (đếm số ngày có precision < 0.75 và recall < 0.75).

4. **Tự động hóa quy trình báo cáo**

   - Tạo thư mục `project/reports`.
   - Kết hợp các lệnh, lưu kết quả vào `project/reports/weekly.txt`.
   - Tạo file dữ liệu (`region_counts.dat`) cho biểu đồ.
   - Dùng `gnuplot` vẽ biểu đồ người dùng theo khu vực (`users_by_region.png`).

16 Tóm tắt và Tổng kết phần 3