I'll break down `cut`, `sort`, `uniq`, and `wc` in detail, covering their syntax, options, and use cases with in-depth explanations of their operations.

🔹 1. The `cut` Command

The `cut` command is used to **extract specific parts of each line** from a file or standard input. It is commonly used to **select columns, extract fields, or split data**.

1️⃣ Basic Syntax

cut -d 'delimiter' -f field_number file

- `-d 'delimiter'` → Specifies the **delimiter** (default is **Tab**).
- `-f field_number` → Extracts **specific field(s)**.
- `file` → Input file.

2️⃣ Extracting Columns (Fields)

Example: Extract the 2nd Column from a CSV File

cut -d ',' -f 2 data.csv

✅ Extracts the **2nd field** from `data.csv`, assuming **comma-separated values (CSV)**.

🔹 **How It Works?**

Name	Age	Country
John	25	USA
Alice	30	UK

💡 Output:

25
30

3️⃣ Extracting Multiple Fields

cut -d ',' -f 1,3 data.csv

✅ Extracts **1st and 3rd** fields.

💡 Output:

John,USA
Alice,UK

4️⃣ Cutting by Character Positions (`-c`)

cut -c 1-5 names.txt

✅ Extracts **characters 1 to 5** from each line.

💡 Input:

Jonathan
Michael

💡 Output:

Jonat
Micha

5️⃣ Cutting Without a Delimiter (Fixed Width)

cut -c 3- names.txt

✅ Extracts **everything from the 3rd character onwards**.

🔹 2. The `sort` Command

The `sort` command arranges **lines of text files** in **ascending** or **descending** order.

1️⃣ Basic Syntax

sort [options] file

- By default, `sort` sorts **alphabetically**.

2️⃣ Sorting a File Alphabetically

sort names.txt

✅ Sorts **names in ascending order**.

💡 Input:

banana
apple
cherry

💡 Output:

apple
banana
cherry

3️⃣ Sorting in Reverse Order (`-r`)

sort -r names.txt

✅ Sorts in **descending** order.

💡 Output:

cherry
banana
apple

4️⃣ Sorting Numerically (`-n`)

sort -n numbers.txt

✅ Sorts **numbers** correctly.

💡 Input:

10
2
30
5

💡 Output:

2
5
10
30

5️⃣ Sorting by Column (`-k`)

sort -t ',' -k2 -n data.csv

✅ Sorts **CSV** data **by the 2nd column numerically**.

💡 Input (`data.csv`):

Alice,30
John,25
Bob,40

💡 Output:

John,25
Alice,30
Bob,40

6️⃣ Sorting Unique Values (`-u`)

sort -u names.txt

✅ Removes **duplicate entries**.

7️⃣ Case-Insensitive Sorting (`-f`)

sort -f names.txt

✅ Ignores case when sorting.

💡 Input:

apple
Banana
cherry

💡 Output:

apple
Banana
cherry

🔹 3. The `uniq` Command

The `uniq` command **removes duplicate adjacent lines**.

1️⃣ Basic Syntax

uniq [options] file

- **`uniq` only works if duplicates are consecutive.**
- Always **sort** before using `uniq`.

2️⃣ Removing Duplicate Lines

sort names.txt | uniq

✅ Sorts and removes duplicate lines.

💡 Input:

apple
apple
banana
banana
cherry

💡 Output:

apple
banana
cherry

3️⃣ Counting Duplicate Occurrences (`-c`)

sort names.txt | uniq -c

✅ Counts **repeated lines**.

💡 Output:

2 apple
2 banana
1 cherry

4️⃣ Display Only Duplicates (`-d`)

sort names.txt | uniq -d

✅ Shows **only repeated lines**.

💡 Output:

apple
banana

5️⃣ Display Only Unique (Non-Repeated) Lines (`-u`)

sort names.txt | uniq -u

✅ Shows **only unique lines** (appearing once).

💡 Output:

cherry

🔹 4. The `wc` Command

The wc (Word Count) command **counts the number of lines, words, and characters** in a file.

1️⃣ Basic Syntax

wc [options] file

2️⃣ Counting Lines (`-l`)

wc -l data.txt

✅ Counts the **number of lines** in `data.txt`.

💡 Output:

5 data.txt

(Means 5 lines)

3️⃣ Counting Words (`-w`)

wc -w data.txt

✅ Counts **words**.

4️⃣ Counting Characters (`-c`)

wc -c data.txt

✅ Counts **characters**.

5️⃣ Counting Bytes (`-m`)

wc -m data.txt

✅ Counts **characters, including special ones** (like Unicode).

🔹 5. Combining Commands

1️⃣ Find the Most Frequent Word in a File

cat file.txt | tr ' ' '\n' | sort | uniq -c | sort -nr | head -10

✅ **Breakdown:**

1. `cat file.txt` → Read the file.
2. `tr ' ' '\n'` → Convert spaces to new lines (one word per line).
3. `sort` → Sort words alphabetically.
4. `uniq -c` → Count occurrences.
5. `sort -nr` → Sort counts in descending order.
6. `head -10` → Show **top 10 words**.

Conclusion

- **`cut`** → Extracts specific **columns/characters**.
- **`sort`** → Sorts data **alphabetically, numerically, or by column**.
- **`uniq`** → Removes **duplicate** adjacent lines.
- **`wc`** → Counts **lines, words, characters**.

Would you like more examples or deeper breakdowns? 😊

I'll now break down these options in **depth** with detailed explanations and **step-by-step illustrations**.

🔹 1. `--complement` Option

The **`--complement`** option is used with the `cut` command to **invert the selection**—instead of extracting the specified fields, it **excludes them**, displaying everything **except** what is selected.

🔸 1.1 Basic Syntax

cut --complement -f <fields> -d '<delimiter>' <file>

- **`-f <fields>`** → Specifies which fields (columns) to extract or exclude.

- `-d '<delimiter>'` → Defines the field separator (default is **TAB**).
- `--complement` → **Inverts** the selection (removes the specified fields instead of keeping them).

🔸 1.2 Example Use Case

Let's say we have a **CSV file** called `data.csv`:

File (`data.csv`):

ID,Name,Age,Gender,Salary
1,Alice,25,F,50000
2,Bob,30,M,60000
3,Charlie,35,M,70000

1️⃣ Extracting Only Certain Fields (`-f`)

If we **extract** columns **1 (ID) and 3 (Age)**:

cut -d ',' -f 1,3 data.csv

✅ Output:

ID,Age
1,25
2,30
3,35

2️⃣ Using `--complement` to Remove Fields

Instead of **keeping** fields 1 and 3, let's **remove** them:

cut --complement -d ',' -f 1,3 data.csv

✅ Output:

Name,Gender,Salary
Alice,F,50000

Bob,M,60000
Charlie,M,70000

💡 **What happened?**

- **Without `--complement`**, `cut` keeps fields 1 and 3.
- **With `--complement`**, it removes fields **1 and 3**, keeping everything else.

3️⃣ Practical Use Case: Filtering Out Unnecessary Data

Suppose you need to process a **large CSV** but don't want to include **ID and Age** in your report. Instead of manually selecting all other fields, you can just **exclude** them using `--complement`.

🔹 2. `--output-delimiter` Option

The `--output-delimiter` option lets you specify a **custom delimiter** instead of using the default one.

🔸 2.1 Basic Syntax

cut -d '<input-delimiter>' -f <fields> --output-delimiter='<new-delimiter>' <file>

- `-d '<input-delimiter>'` → Specifies the delimiter used in the input file.
- `-f <fields>` → Specifies which fields to extract.
- `--output-delimiter='<new-delimiter>'` → Changes the delimiter **in the output**.

🔸 2.2 Example Use Case

Let's take the same `data.csv` file:

File (`data.csv`):

ID,Name,Age,Gender,Salary
1,Alice,25,F,50000
2,Bob,30,M,60000
3,Charlie,35,M,70000

1️⃣ Extracting Fields with Default Output Delimiter

cut -d ',' -f 1,2,5 data.csv

✅ Output:

ID,Name,Salary
1,Alice,50000
2,Bob,60000
3,Charlie,70000

- The output **preserves** the input delimiter (`,`).

2️⃣ Using `--output-delimiter` to Change the Delimiter

Now, let's replace `,` with `|` (pipe symbol):

cut -d ',' -f 1,2,5 --output-delimiter='|' data.csv

✅ Output:

ID|Name|Salary
1|Alice|50000
2|Bob|60000
3|Charlie|70000

💡 What changed?

- Input delimiter (`-d ','`) was a **comma**.
- Output delimiter (`--output-delimiter='|'`) became a **pipe (`|`)**.

3️⃣ Why Use `--output-delimiter`?

- If you're **converting** between formats (e.g., CSV → TSV).
- When working with **inconsistent data formats**.
- If you need **cleaner** output for further processing.

🔹 Summary

Option	Purpose	Example Usage
`--complement`	Excludes instead of selecting fields	`cut --complement -d ',' -f 1,3 data.csv`
`--output-delimiter`	Changes the delimiter in the output	`cut -d ',' -f 1,2 --output-delimiter='

Would you like more advanced examples or a deeper breakdown? 😊