The `tr`, `sort`, and `uniq` Commands in Linux/Unix

These three commands are fundamental text-processing utilities from GNU coreutils (part of virtually every Linux/Unix system). They are designed to work efficiently with streams (standard input/output) and are frequently piped together for powerful text manipulation tasks.

- `tr` – Translates, deletes, or squeezes characters.
- `sort` – Sorts lines of text.
- `uniq` – Filters adjacent duplicate lines (often used after `sort` to handle global duplicates).

Below is an exhaustive breakdown of each command, covering synopsis, description, every option, special features (like character classes or key definitions), quirks, standards notes, and numerous examples. I'll conclude with common pipelines that combine them.

1. `tr` – Translate or Delete Characters

Synopsis

```
text
tr [OPTION]... SET1 [SET2]
```

Description

`tr` copies standard input to standard output, performing one or more operations:

- **Translation**: Replace characters in SET1 with corresponding characters in SET2.
- **Deletion**: Remove characters listed in SET1.
- **Squeezing**: Replace sequences of repeated characters (from the last specified set) with a single instance.

Operations are applied in order: deletion → translation → squeezing.

SET1 and SET2 are character sets. Special constructs:

- `CHAR1-CHAR2` : Range of characters (inclusive, ascending).
- `[CHAR*]` : In SET2, repeat CHAR to match length of SET1.
- `[CHAR*REPEAT]` : Repeat CHAR exactly REPEAT times (REPEAT can be octal if starting with 0).
- Escapes: `\\` , `\a` (bell), `\b` , `\f` , `\n` , `\r` , `\t` , `\v` , `\NNN` (octal).

Character Classes (expand in unspecified order)

- `[:alnum:]` – Letters and digits
- `[:alpha:]` – Letters
- `[:blank:]` – Horizontal whitespace (space, tab)
- `[:cntrl:]` – Control characters
- `[:digit:]` – Digits (0-9)
- `[:graph:]` – Printable non-space characters
- `[:lower:]` – Lowercase letters
- `[:print:]` – Printable characters (including space)
- `[:punct:]` – Punctuation
- `[:space:]` – Horizontal/vertical whitespace
- `[:upper:]` – Uppercase letters
- `[:xdigit:]` – Hex digits (0-9, a-f, A-F)
- `[=CHAR=]` – All characters equivalent to CHAR (useful for equivalence classes in some locales)

All Options

- `-c`, `-C`, `--complement`: Use the complement of SET1 (all characters not in SET1).
- `-d`, `--delete`: Delete characters in SET1 (no translation).
- `-s`, `--squeeze-repeats`: Squeeze repeated characters from the last specified set.
- `-t`, `--truncate-set1`: Truncate SET1 to the length of SET2 before translation.
- `--help`, `--version`: Standard help/version.

Quirks & Notes

- Full multi-byte character support is only guaranteed in single-byte ("safe") locales. In UTF-8 or other multi-byte locales, unexpected behavior can occur. Workaround: Prefix with `LC_ALL=C tr ...` for byte-level processing.
- When translating, SET2 is extended by repeating its last character if shorter than SET1. Excess characters in SET2 are ignored.
- Case conversion is commonly done with `[:lower:]` and `[:upper:]` pairs.
- Squeezing happens after translation/deletion.

Examples

```
Bash

# Convert lowercase to uppercase
echo "hello world" | tr '[:lower:]' '[:upper:]'
# Output: HELLO WORLD

# Delete all digits
echo "abc123def456" | tr -d '[:digit:]'
# Output: abcdef

# Squeeze multiple spaces into one
echo "a   b    c" | tr -s ' '
# Output: a b c

# Replace spaces with newlines (word per line)
echo "one two three" | tr ' ' '\n'
# Output:
# one
# two
# three

# Complement: delete everything except vowels
echo "hello world" | tr -d -c 'aeiou'
# Output: eo o

# Using repeats
tr 'a' '[x*10]' <<< "aaaaa"   # Replaces each 'a' with 10 x's (if lengths mat

# ROT13 cipher (classic)
echo "hello" | tr 'A-Za-z' 'N-ZA-Mn-za-m'
# Output: uryyb
```

2. `sort` – Sort Lines of Text Files

Synopsis

```
text

sort [OPTION]... [FILE]...
# Or: sort [OPTION]... --files0-from=F
```

Description

Sorts lines from FILE(s) (or stdin) and writes to stdout. Concatenates files before sorting. Without files or with `-`, reads stdin.

Key Specification (`-k` or `--key=KEYDEF`)

KEYDEF format: `F[.C][OPTS][,F[.C][OPTS]]`

- F = field number (1-based)
- C = character position in field (1-based)
- OPTS = single-letter modifiers (`b d f g i M h n R r V`) overriding global settings
  If no key, entire line is used. `--debug` highlights used portions.

All Options (grouped)

Ordering options

- `-b` , `--ignore-leading-blanks`
- `-d` , `--dictionary-order` (only blanks + alphanumerics)
- `-f` , `--ignore-case`
- `-g` , `--general-numeric-sort`
- `-i` , `--ignore-nonprinting`
- `-M` , `--month-sort`
- `-h` , `--human-numeric-sort` (e.g., 2K > 1G)
- `-n` , `--numeric-sort`
- `-R` , `--random-sort` (shuffle, group identical keys)
- `-r` , `--reverse`
- `-V` , `--version-sort` (natural version numbers)
- `--sort=WORD` (explicitly choose type)

Other options

- `--batch-size=NMERGE` – Limit simultaneous merges
- `-c`, `--check` / `-C` – Check if already sorted (silent with `-C`)
- `--compress-program=PROG` – Use external compressor for temp files
- `--debug` – Highlight sort keys and warn on issues
- `--files0-from=F` – Read NUL-terminated filenames from F
- `-k`, `--key=KEYDEF`
- `-m`, `--merge` – Merge pre-sorted files
- `-o`, `--output=FILE`
- `-s`, `--stable` – Disable last-resort comparison
- `-S`, `--buffer-size=SIZE` (suffixes: K, M, G, etc.)
- `-t`, `--field-separator=SEP`
- `-T`, `--temporary-directory=DIR`
- `--parallel=N`
- `-u`, `--unique` – Output only first of equal lines
- `-z`, `--zero-terminated`
- `--help`, `--version`

Quirks & Notes

- Sort order is heavily affected by locale (LC_ALL/LC_COLLATE). For traditional byte-order (ASCII), use `LC_ALL=C sort ...`.
- `-u` combines sorting and unique filtering (stable).
- `--stable` (`-s`) disables full-line comparison when keys are equal (faster, preserves input order for equal keys).
- Human/version/month sorts handle special formats intelligently.

Examples

```bash
# Basic alphabetic sort
sort file.txt

# Numeric sort
sort -n numbers.txt

# Reverse numeric
sort -nr scores.txt

# Sort by second field (space-separated)
sort -k2 file.txt

# Human-readable sizes (largest first)
ls -lh | sort -hr -k5

# Version sort
echo -e "v1.10\nv1.2\nv1.1" | sort -V
# Output: v1.1\nv1.2\nv1.10

# Stable sort on key, then full line
sort -s -k1,1 file.txt
```

3. `uniq` – Report or Omit Repeated Lines

Synopsis

```text
uniq [OPTION]... [INPUT [OUTPUT]]
```

Description

Filters adjacent duplicate lines. First occurrence is kept by default. Non-adjacent duplicates are not detected → usually pipe from `sort`. Fields are runs of blanks followed by non-blanks.

All Options

- `-c`, `--count` : Prefix with occurrence count
- `-d`, `--repeated` : Print only duplicate lines (one per group)
- `-D` : Print all duplicate lines
- `--all-repeated[=METHOD]` : Like `-D`; METHOD = none/prepend/separate
- `-f N`, `--skip-fields=N` : Ignore first N fields
- `--group[=METHOD]` : Show all lines, separate groups with blank line (separate/prepend/append/both)
- `-i`, `--ignore-case`
- `-s N`, `--skip-chars=N` : Ignore first N characters
- `-u`, `--unique` : Print only unique lines
- `-w N`, `--check-chars=N` : Compare only first N characters
- `-z`, `--zero-terminated`
- `--help`, `--version`

Quirks & Notes

- Only adjacent duplicates are processed – always sort first for global uniqueness.
- `sort -u` is often a faster alternative (single process).
- `-c` counts are right-aligned with space padding.

Examples

```bash
# Remove adjacent duplicates
sort file.txt | uniq

# Count occurrences
sort file.txt | uniq -c

# Only unique lines (ignore duplicates)
sort file.txt | uniq -u

# Only duplicated lines
sort file.txt | uniq -d

# Ignore case
uniq -i file.txt

# Skip first field when comparing
uniq -f1 file.txt
```

Common Pipelines Combining `tr`, `sort`, and `uniq`

These commands shine when chained:

1. **Word frequency count** (top 10 most common words, case-insensitive)

```bash
cat file.txt \
  | tr -cs '[:alnum:]' '\n' \     # Split into words, keep only alnum
  | tr '[:upper:]' '[:lower:]' \ # Lowercase
  | sort \
  | uniq -c \
  | sort -nr \                   # Numeric reverse (highest first)
  | head -10
```

1. **Unique sorted lines**

```bash
sort file.txt | uniq
# Or simply: sort -u file.txt
```

1. **Count duplicate lines exactly**

```
Bash
sort file.txt | uniq -c | sort -nr
```

1. **Extract and count domains from URLs**

```
Bash
cat urls.txt | tr ' ' '\n' | cut -d/ -f3 | sort | uniq -c | sort -nr
```

1. **Remove all punctuation and count words**

```
Bash
cat text.txt | tr -d '[:punct:]' | tr -s '[:space:]' '\n' | sort | uniq -c
```

Key Pipeline Quirks

- `uniq` requires sorted input for meaningful results.
- `sort -u` vs `sort | uniq`: `sort -u` is generally faster and stable; `sort | uniq` is more portable to very old systems.
- Use `LC_ALL=C` prefix for any of these when dealing with non-ASCII text to avoid locale surprises.

These commands are lightweight, extremely fast, and foundational for shell scripting and one-liners. Master them, and you'll handle most text-processing tasks without heavier tools!