Common Pipeline Utilities

grep searches the named input FILEs (or standard input if no files are named, or the file name - is given) for lines containing a match to the given PATTERN. By default, grep prints the matching lines. It stands for "global regular expression print."

```
grep [options] PATTERN [FILE...]
grep [options] [-e PATTERN | -f FILE] [FILE...]
```

-e is the flag for indicating the pattern you want to match against.

-E allows you to use Extended Regular Expressions (rather than basic).
In basic regular expressions, the meta-characters ?, +, {, |, (, and) lose their special meaning; instead use the backslashed versions \?, \+, \{, \|, \(, and \).

-f Lets you obtain patterns from FILE, one per line. The empty file contains zero patterns, and therefore matches nothing.

Use the --color option to highlight matches within their lines.

-n prefixes each matching line with the line number.

-i specifies to perform case-insensitive matching.

-r extends our search recursively to subdirectories and any files that they contain.

---

head makes it easy to output the first part of files.

Prints the first 10 lines of each file to standard output by default.
If more than one file, identifies each section of output with a header that contains the filename.
If no file or - specified, head reads from standard input. For example:

find | head

head -n filename will print the first n lines:
head -5 myfile.py

head -c n file prints the first n bytes of the file:
head -c 100 myfile.py

-q means never print headers, -v means always print headers.

tail outputs the last lines of files.
It is a lot like head with -n, -c, -q, and -v options
By default, it prints the last 10 lines to standard out.

The -f option makes tail output appended data as the file grows.
The -s option sets the refresh rate for -f (1 second by default).

Example:

```
tail -f access.log | grep 24.10.160.10
```

This is a useful example of using tail and grep to selectively monitor a log file in real time.
In this command, tail monitors the file `access.log` and pipes the final ten lines, and any new lines added, to grep. grep reads the output from tail, and outputs only those lines which contain the IP address 24.10.160.10.

---

sort sorts the contents of a file line by line.

By default:

- lines starting with a number will appear before lines starting with a letter;
- lines starting with a letter that appears earlier in the alphabet will appear before lines starting with a letter that appears later in the alphabet;
- lines starting with a lowercase letter will appear before lines starting with the same letter in uppercase.

-r reverses the default sorting
-R randomly shuffles the lines
-u shows only one of each line in the sorted list (no duplicate lines. unique).
There are lots of other options

Note that sort doesn't change the file. It just prints the sorted version to standard out.

```
sort unsorted.txt
```

You can redirect the output as usual:

```
sort unsorted.txt > sorted.txt
```

However, this WILL NOT WORK with the same file:

```
sort file.txt > file.txt
```

Because the shell truncates the contents of the file before sort gets a chance to read it. Instead, use:

```
sort file.txt | tee file.txt
```

tee reads from standard input and writes to standard output and to files at the same time. It is named after a T-splitter in plumbing because the output goes to two places at once.

The -a option appends to files instead of overwriting.

---

uniq, by default, filters out adjacent, matching lines from the input file, writing the filtered data to standard out. If no options are specified, matching adjacent lines are merged to the first occurrence.

-c prefixes lines with a number representing how many times they occurred.
-d will only print duplicated lines
-i ignores case
-u only prints unique lines

NOTE: uniq does not detect repeated lines unless they are adjacent. You may want to sort the input first, or use sort -u instead of uniq.

---

tr automatically translates (substitutes, or maps) one set of characters to another. It copies the standard input to the standard output with substitution or deletion of selected characters.

```
tr [Options] set1 [set2]
```

If both set1 and set2 are specified and '-d' OPTION is not specified, then tr will replace each character of the input in set1 with the character in the same position in set2. If set1 is longer than set2, the last character found in set2 is duplicated until set1 is exhausted.

Lowercase to uppercase:

```
tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
tr [:lower:] [:upper:]
tr a-z A-Z
```

-d deletes characters in set1 from the input.
-C takes the compliment of set1 (all characters not in set1)
-s Squeezes multiple occurrences of the characters listed in the last operand (either set1 or set2) in the input into a single instance of the character. This occurs after all deletion and translation is completed.

Convert multiple adjacent spaces into a single space:

```
tr -s [:space:] ' '
```

Remove non-printable characters from a file:

```
tr -cd [:print:] < file.txt
```

Join all the lines in a file into a single line:

```
tr -s '\n' ' ' < file.
```