cat -n	display lines with line numbers (all lines are numbered)
head -n N	output the first N lines of each file
tail -n N	output the last N lines of each file
tail -n +N	output the end of each file beginning with line N (from the beginning)
grep "pattern"	output lines that contain pattern (no options) **-i** - contain pattern, ignoring case **-v** - dont contain pattern **-c** - count the number of lines that contain pattern **-l** - output the name of the files that contain at least one match
sort	sort lines. default: alphabetic, smallest to largest; use -n for numeric sort; use -r for reverse order (largest to smallest)
type	output information about where a command comes from (also useful, but more limited in bash: whereis and which)
more	display files page by page. Commands to control display: space - forward one page enter - forward one line ^B - back one page **/pattern** - search forward to next instance of pattern **q** quit
ls	**-L** - follow symlinks **-lu** long format sorted by modification date **-i** - show inode numbers **-d** - list directories themselves not contents
ln orig new	create a second link (named **new**) to the file that **orig** refers to default - create hard link Use **-s** to create symbolic link
man	get information from the Unix manual **man x** output the first manual page found that is named **x** **man -k yy** output the names of pages whose terse description contains **yy** **man N yy** limit the search to section N of the Unix manual
find dir [opts]	output the path to each thing beneath directory **dir** that matches the options opts: **-type x** find objects of type **x**, where **x** is **f**(file), **d**(dir), **l**(symlink) **-name 'pat'** whose name matches the given wildcard pattern **pat** (**pat** must be quoted) **-L** If a symlink is found, follow it and examine what it refers to. (needs other options (such as **-type f**) to be interesting.)
locate "pattern"	**locate** searches a database of all files on the system looking for patterns in filenames. The database is updated daily (usually). **pattern** is a wildcard (globbing) pattern. If the pattern does not contain wildcards, leading and trailing asterisks are assumed (***pattern***) If the option **--regexp** is used, **pattern** is basic regexp If the option **--regex** is used, **pattern** is extended regexp
tee [-a] file	Duplicate data going to standard output. Data still goes to standard output, but it is also copied to **file**. **grep 'abc' foo | tee abcs** would display the matches on stdout *and* save them in the file **abcs** **-a**: append **file** instead of overwriting it.
uniq	remove adjacent duplicate lines. Unique lines are sent to stdout.

sort

general options

-tx use **x** as delimiter. Whitespace is the default.

-f fold (ignore) case

-u (unique) - remove lines with duplicate keys

-o outfile put the output file in **outfile** instead of stdout

sort keys

-kstart[,stop] **start** and **stop** are **f[.c]**

the modifiers **r** (reverse) and **n** (numeric) can be appended to the key specifier

Interpretation of keys. Note that [bracketed parts] are optional

- the first character of the key is at field **f** and character offset **c** of **start**
- the last character of the key is at field **f** and character offset **c** of **stop**.
- If **.c** is missing on start, it means the first character of the field.
- If **.c** is missing on stop, it means the last character of the field.
- If **,stop** is missing, it means the last field of the record.
- Remaining ties are broken by sorting the unsorted fields alphabetically starting with field 1

Examples

Given a colon-delimited file **data** *of four fields*

```
sort -t: -k2,2n data
```

the key is field 2 only, numeric. Ties are broken by sorting the ties on field 1 (alphabetic) then on field 3 (alphabetic) then on field 4 (alphabetic)

```
sort -t: -k2n data
```

the key is field 2 numeric, but this key also includes fields 3 and 4 (numeric) to break ties. If any remaining ties exist (two records which are identical in fields 2-4), field 1 is sorted (alphabetic) to break them.

```
sort -t: -k4 -u data
```

the key is field 4, which is sorted alphabetically. Then the rest of the line is sorted to break any remaining ties. Since **-u** is used, only the first record that has a particular value in field 4 is kept - records with duplicate field 4's are deleted.

```
sort -t: -f -k2,2n -k4 data
```

the first key is field 2 numeric. Ties are broken using the second key, which is field 4 alphabetic. Remaining ties are broken by sorting on fields 1 (alphabetic) then on field 3 (alphabetic). All sorts ignore case.

```
sort -t: -o data -k3.1,3.1n -k2r data
```

the first key is a single character - the first character of field 3. It is a numeric sort, which is redundant for a single digit. Ties are broken by sorting alphabetically in reverse order by field 2, then by the rest of field 3, then by field 4 then by field 1.

The output of this sort is placed in a file named **data**, replacing the original file. This moves the responsibility of creating the output file to the **sort** command so that the input file can be read before the output file is created. The command below overwrites the input file before the **sort** command starts, destroying the file in the process and resulting in an empty file.

```
sort -t: -k3.1,3.1n -k2r data > data   # DONT DO THIS!!!!!
```