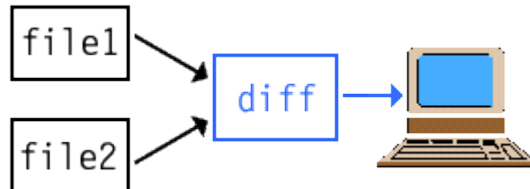


diff

- ◆ It is common, during development, to have several versions of a file around.
- ◆ *diff* allows you to quickly determine how the latest version of the file differs from a previous version



- ◆ *diff* works with text files only; *cmp* works with all types of files.

Example diff Usage

| | |
|---|---|
| <pre>\$ cat FILE.OLD Mats Sundin 13 Doug Gilmour 93 C Wendel Clark 17 A Mike Palmateer 29 \$ cat FILE.NEW Mats Sundin 13 C Wendel Clark 17 A Mike Palmateer 29 Tie Domi 28 \$</pre> | <pre>\$ diff FILE.OLD FILE.NEW 1,2c1 <Mats Sundin 13 <Doug Gilmour 93 C - - - - - >Mats Sundin 13 C 4a4 >Tie Domi 28 \$</pre> |
|---|---|

sort

- ◆ Sorts files in alphabetic or numeric order.
- ◆ The default sort key is an entire line.
- ◆ The sort key can be restricted to a particular field of each line by using options.
- ◆ Output is sent to *STDOUT*.

sort Examples

- ◆

```
$ cat FILE1
Mats Sundin 13 C
Wendel Clark 17 A
Felix Potvin 29
Tie Domi 28
```

```
$ sort FILE1
Mats Sundin 13 C
Mike Palmateer 29
Tie Domi 28
Wendel Clark 17 A
```

- ◆ Each line of text is an object. The objects are sorted into alphabetical order.
- ◆ Suppose we wish to sort on last names:

```
$ sort +1 FILE1
Wendel Clark 17 A
Tie Domi 28
Mike Palmateer 29
Mats Sundin 13 C
```

- ◆ The `+1` option means skip the first field, then treat the rest of the line as an object.
- ◆ Suppose we wish to sort on numbers:

```
$ sort +2n FILE1
Mats Sundin 13 C
Wendel Clark 17 A
Tie Domi 28
Mike Palmateer 29
```

- ◆ The `+2n` option selects the third field and sorts numerically.

More sort Examples

| | |
|--|--|
| <code>sort +1 -2 FILE4</code> | Skip 1 field before sorting and stop after sorting the second field. |
| <code>sort -b +2 FILES</code> | Skip 2 fields, then sort. Ignore leading blanks. |
| <code>sort +2n +1 FILE1.TXT</code> | Sort the third field numerically, then for all lines with identical third fields, order according to the second field (<i>double sort</i>). |
| <code>sort -t: +2n /etc/passwd</code> | Sort the third field numerically, and the <i>field separator</i> is a colon (not a space). |
| <code>sort FILE1 FILE2 > OUTFILE</code> | Sort the combined input of both files and store the results in <i>OUTFILE</i> . |
| <code>sort -m FILE1 FILE2 > OUTFILE</code> | <i>FILE1</i> and <i>FILE2</i> are already sorted; merge them into one big sorted file named <i>OUTFILE</i> . |
| <code>sort -t: +3n -4 +0 -1 /etc/passwd</code> | Sort the fourth field numerically, and the <i>field separator</i> is a colon (not a space). In the case of identical fourth fields, sort by the first field. |
| <code>sort +2n -3 +1 -2 FILE1.TXT</code> | Sort the third field numerically, then for all lines with identical third fields, order according to the second field (<i>double sort</i>). |
| <code>sort +2.10 FILE2.TXT</code> | Sort, starting at the 11th character the third field. |
| <code>sort +2.10 -2.15 FILE2.TXT</code> | Sort, with the key starting at the 11th character the third field, and ending at the 16th character of the third field. |

tr - Translate Characters

- ◆ The following will reverse the alphabet for all lower-case letters in *FILE3*:

```
tr "abcdefghijklmnopqrstuvwxyz" \  
  "zyxwvutsrqponmlkjihgfedcba" < FILE3
```

- ◆ To make a file all upper case:

```
tr '[a-z]' '[A-Z]' < FILE4.DAT
```

- ◆ To get rid of all numbers in *FILE4* (*delete*):

```
tr -d "[0-9]" < FILE4
```

tr Examples

- ◆ Use *complement-delete* to get rid of everything except digits from *DATA.4*:

```
tr -cd "[0-9]" < DATA.4
```

- ◆ To get rid of everything except digits and newlines:

```
tr -cd '\012[0-9]' < FILE7.DATA
```

wc - Word Count

SYNTAX:

```
wc [-lwc] FILE FILE ...
```

DESCRIPTION:

wc counts words, lines, and characters. A word is a string of characters delimited by spaces, tabs, or newlines.

- ◆ If any options are used, just those counts appear.
- ◆ If no file is specified, input will be taken from *STDIN*.

grep

- ◆ Search a file for lines matching a limited **regular expression**.
- ◆ Any line of the file that has text that matches is sent to *STDOUT*.
- ◆ Lines with no match are filtered out, and don't show on *STDOUT*.
- ◆ You have to use single quotes (') if you are using *shell metacharacters* in your search pattern. Single quotes cause the shell to ignore what's inside and pass the pattern directly to *grep*.

grep Examples

- ◆ The general syntax is:

```
grep PATTERN FILE ...
```
- ◆ To search a document for all lines containing the word "GRAPHICS":

```
grep GRAPHICS some.file
```
- ◆ To search all files that end in ".TXT" for references to a particular hockey player:

```
grep "Mario Lemieux" *.TXT
```

Advanced Patterns

| | |
|--------------------------------|---|
| . | Matches any character. |
| [<i>string</i>] | Matches any character in <i>string</i> . |
| [^<i>STRING</i>] | Matches any character not in <i>STRING</i> . |
| ^ | Anchors to the beginning of the line. |
| \$ | Anchors to the end of the line. |
| <i>some pattern</i>* | Matches zero or more occurrences of <i>some pattern</i> , which may be a single character, or a wildcard. |

grep Patterns

- ◆ Care should be taken when the following characters are used in a *grep* pattern, as they are meaningful to the shell:

`$ * [^ | ? ' " () & \] < >`

- ◆ It is safest to include the entire pattern in single quotes `' '`.
- ◆ If the pattern contains any spaces, it must be enclosed in single or double quotes `" "`.

Advanced grep Examples

- ◆ To find all lines with words ending in `'ing'`:

```
grep '[a-zA-z]*ing' text.file
```

- ◆ To find all lines containing parenthesized numbers:

```
grep '([0-9]*)' FILE3.DAT
```

- ◆ To find all *nroff* commands:

```
grep '^\. [a-zA-z]' this.file
```

egrep

- ◆ Member of the *grep* family of commands, along with *grep* and *fgrep*.
- ◆ Allows more extensive pattern matching than *grep*.
- ◆ Is slower than *grep*.

egrep Patterns

| Wildcard | Meaning |
|--|--|
| (<i>PATTERN</i>) | Match <i>PATTERN</i> - can be arbitrarily complex. |
| <i>PATTERN</i>* | Match zero or more occurrences of <i>PATTERN</i> . |
| <i>PATTERN</i>+ | Match one or more occurrences of <i>PATTERN</i> . |
| <i>PATTERN</i> <i>PATTERN</i> | Match either <i>PATTERN</i> . |

egrep Examples

- ◆ To find every mention of Canada or Mexico:

```
egrep 'Canada|Mexico' FILE2
```
- ◆ To print all headings of the form "*N.N.N...N HEADING*":

```
egrep '[0-9]+(\.[0-9]+)*[ ]+[a-zA-Z]+'
```
- ◆ To print all lines with reference to a certain television program:

```
egrep 'Beavis|Butthead' FILE6.NAMES
```