Reference Section

- &&
- ¦¦
- $0
- $1, $2 ... $9
- $#
- ${#VAR}
- $?
- $*
- $IFS
- $RANDOM
- $SECONDS
- $( )
- (( )), $(( ))
- /dev/null
- 2> filename
- 2>&1
- #!
- Arrays
- backslash
- bc
- break
- Builtin
- cal
- case
- cat
- Comments
- continue
- cut
- date
- Double-quotes
- echo
- eval
- exit
- exit-code
- expr
- false
- Filename expansion
- for
- getopts
- grep
- if
- nl
- pipe
- read
- Run-quotes
- sed
- set
- Shell functions
- shift
- sleep
- stty
- test
- tput
- tr
- true
- typeset
- unset
- Variables
- wc
- while
- whitespace

&&

&&

The && pair are effectively a one line *if* statement with the syntax "cmd1 && cmd2". The command to the left of && is executed and if it exits with an *exit-code* of zero, then the right-hand command is also executed. The left-hand command is typically a *test* statement:

```
[ "$END" = "Y" ] && exit 0
```

if this command exits with exit-code zero — then — run this command

```
cmd1 && cmd2
```

is equivalent to

```
cmd1
if [ $? -eq 0 ]
then
    cmd2
fi
```

and

```
if cmd1
then
    cmd2
fi
```

Whilst the && syntax is limited by not supporting an "*else*" statement or multiple lines of code as *if* does, it is very useful for condensing one line, simple *if* statements.

```
if [ "$MIN" -lt 0 ]
then
    MIN=0
fi

if [ -z "$DAY" ]
then
    DAY=`date "+%a"`
fi
```

can be condensed to

```
[ "$MIN" -lt 0 ] && MIN=0

[ -z "$DAY" ] &&
    DAY=`date "+%a"`
```

Note that the && syntax allows a newline after "&&". This allows the code to be laid out in a more readable format as && statements can get very long.

See also: || , test, if

continue

```
continue [n]
```

Continue is used as a control mechanism for the *for* and *while* loops. When a *continue* is called within a loop, the execution of the loop is immediately returned to the beginning.

When called within a *while loop*, the *test* statement is re-evaluated prior to the loop being restarted (i.e. if the *test* statement was true) and when called within a *for* loop, the loop is restarted with the next word from the list:

```
CNT=0
while [ $CNT -le 10 ]
do
    (( CNT += 1 ))

    [ $CNT -eq 5 ] && continue

    echo "$CNT"
done

echo "Finished"
```

```
for CNT in 1 2 3 4 5 6 7 8 9 10
do
    [ $CNT -eq 5 ] && continue

    echo "$CNT"
done

echo "Finished"
```

Will both produce the following output

```
1
2
3
4
6          <-- continue called
7
8
9
10
Finished
```

When used with a number, *continue* will cause nested loops to be restarted:

```
for X in a b c d
do
    for Y in 1 2 3
    do
        [ "$X$Y" = "c2" ] &&
            continue 2

        echo "$X$Y"
    done
done
```

Will produce the following output

```
a1
a2
a3
b1
b2
b3
c1
c3          <-- continue called
d1
d2
d3
Finished
```

See also: break

grep

```
          grep "pattern" FILENAME
command ¦ grep "pattern"
          grep -v "pattern" FILENAME
          grep -i "pattern" FILENAME
          grep -c "pattern" FILENAME
          grep -n "pattern" FILENAME
          grep "pattern" FILE1 FILE2 [ FILE3 [ ... ] ]
          grep -l "pattern" FILE1 FILE2 FILE3 [ ... ]
```

Grep is a program that searches through files or piped input looking for matches to the *pattern* specified on the command line. Any lines that contain the *pattern* are printed out. In it's simplest form *grep* will simply look for plain text as specified as the first argument:

```
$ grep "nus" planets
Venus
Uranus
$ cat planets ¦ grep "ar"
Earth
Uranus
```

planets

```
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
Pluto
```

Grep takes a number of command line options that modify it's behaviour. *Grep -v* reverses the matching, so that only lines that do not match the *pattern* are printed. *Grep -i* makes the *pattern matching* case insensitive.

```
$ grep -v "nus"
planets
Mercury
Earth
Mars
Jupiter
Saturn
Neptune
Pluto
```

```
$ grep -i "ur"
planets
Mercury
Saturn
Uranus
```

grep

Grep -c means that *grep* counts the number of lines that contain the *pattern*. Lines that contain multiple instances of pattern are only counted once. *Grep -n* reports the line numbers of any lines matched:

```
$ grep -c "r" planets
6
$ grep "r" planets
Mercury
Earth
Mars
Jupiter
Saturn
Uranus
```

```
$ grep -n "ur" planets
1: Mercury
6: Saturn
```

The various options can be combined:

```
$ grep -ivn "ur" planets
2: Venus
3: Earth
4: Mars
5: Jupiter
8: Neptune
9: Pluto
```

If more than one file is specified on the command line, *grep* will add the filename to any output generated. This allows the user to identify which files contained the matched lines.

dogs

```
Alsatian
Boxer
Greyhound
Lurcher
Whippet
```

```
$ grep "ia" dogs cats
dogs: Alsatian
cats: Persian
cats: Siamese
```

cats

```
English Blue
Burmese
Persian
Siamese
Somali
```

Grep -l will only display the names of any files that contain the pattern.

```
$ grep -l "ia" dogs cats
dogs
cats
```

grep

Pattern matching with grep

The real power of *grep* comes from the *pattern matching* which allows the programmer to be very precise when specifying what to match.

The following *metacharacters* have a special meaning to *grep* (and to *sed*):

```
  ^     - start of line
  $     - end of line
  .     - any single character
  *     - multiple characters
 [ ]    - any character between the square brackets
\{ \}   - a specific number of characters
```

The carat (^) and dollar-sign ($) anchor the search to the start and/or end of the line:

```
$ grep "^S" cats
Siamese
Somali
```

```
$ grep "r$" dogs
Boxer
Lurcher
```

The period (.) means match any single character:

planets

```
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
Pluto
```

```
$ grep "a.u" planets
Saturn
Uranus
$ grep "^.....$" planets
Venus
Earth
Pluto
```

search for "a" then any character then "u"

search for 5 character words

The square brackets ([ ]) indicate that *grep* should match any one of the characters specified within:

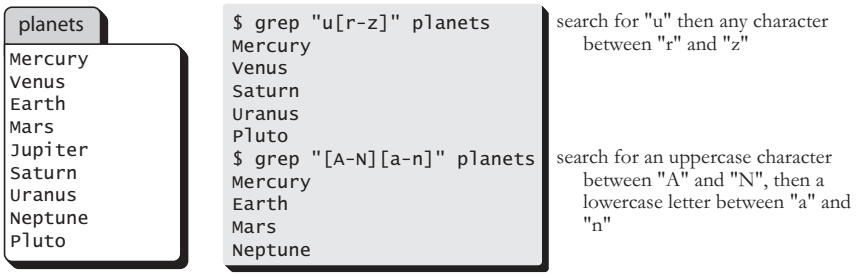
```
$ grep "u[st]" planets
Venus
Uranus
Pluto
$ grep "^[JE]" planets
Earth
Jupiter
```

search for "u" then any charcater from "st"

search for any line starting with one of "JE"

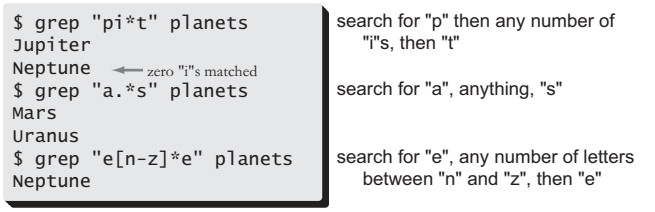
grep

A range of characters can be specified within the square brackets by using a minus:

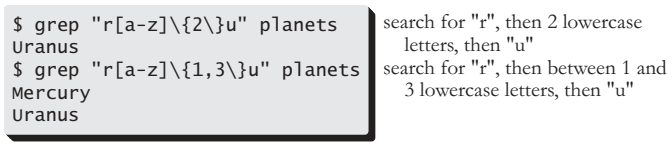


If a minus is to be matched within the square brackets, it must be placed immediately after the open bracket.

The asterisk (*) means "match any number of the previous character". When combined with the period, it means "match anything", and when combined with the square brackets it means "match any number of the characters within the square brackets":



Finally, the "\(\)" pair can be used in place of the asterisk to limit the matches to exactly a number of matches, or between a range of matches:



See also: cat, sed