# Wildcards

Wildcards are useful in many ways for a GNU/Linux system and for various other uses. Commands can use wildcards to perform actions on more than one file at a time, or to find part of a phrase in a text file. There are many uses for wildcards, there are two different major ways that wildcards are used, they are globbing patterns/standard wildcards that are often used by the shell. The alternative is regular expressions, popular with many other commands and popular for use with text searching and manipulation.

> **Tip:** If you have a file with wildcard expressions in it then you can use single quotes to stop bash expanding them or use backslashes (escape characters), or both.
>
> For example if you wanted to create a file called 'fo*' (fo and asterisk) you would have to do it like this (note that you shouldn't create files with names like this, this is just an example):
>
> ```
> touch 'fo*'
> ```

Note that parts of both subsections on wildcards are based (at least in part) off the grep manual and info pages. Please see the *Bibliography* for further information.

## Standard Wildcards (globbing patterns)

Standard wildcards (also known as globbing patterns) are used by various command-line utilities to work with multiple files. For more information on standard wildcards (globbing patterns) refer to the manual page by typing:

```
man 7 glob
```

> **Can be used by:** Standard wildcards are used by nearly any command (including mv, cp, rm and many others).

? (question mark)

> this can represent any *single* character. If you specified something at the command line like "hd?" GNU/Linux would look for hda, hdb, hdc and every other letter/number between a-z, 0-9.

* (asterisk)

> this can represent any number of characters (including zero, in other words, zero or more characters). If you specified a "cd*" it would use "cda", "cdrom", "cdrecord" and *anything* that starts with “cd” also including “cd” itself. "m*l" could by mill, mull, ml, and anything that starts with an m and ends with an l.

[ ] (square brackets)

> specifies a range. If you did m[a,o,u]m it can become: mam, mum, mom if you did: m[a-d]m it can become anything that starts and ends with m and has any character a to d inbetween. For example, these would work: mam, mbm, mcm, mdm. This kind of wildcard specifies an “or” relationship (you only need one to match).

{ } (curly brackets)

terms are separated by commas and each term must be the name of something or a wildcard. This wildcard will copy anything that matches either wildcard(s), or exact name(s) (an “or” relationship, one or the other).

For example, this would be valid:

```
cp {*.doc,*.pdf} ~
```

This will copy anything ending with .doc or .pdf to the users home directory. Note that spaces are not allowed after the commas (or anywhere else).

[!]

This construct is similar to the [ ] construct, except rather than matching any characters inside the brackets, it'll match any character, as long as it is not listed between the [ and ]. This is a logical NOT. For example *rm myfile[!9]* will remove all myfiles* (ie. myfiles1, myfiles2 etc) but won't remove a file with the number 9 anywhere within it's name.

\ (backslash)

is used as an "escape" character, i.e. to protect a subsequent special character. Thus, "\\” searches for a backslash. Note you may need to use quotation marks and backslash(es).

# Regular Expressions

Regular expressions are a type of globbing pattern used when working with text. They are used for any form of manipulation of multiple parts of text and by various programming languages that work with text. For more information on regular expressions refer to the manual page or try an online tutorial, for example IBM Developerworks using regular expressions. For the manual page type:

Type:

```
man 7 regex
```

**Regular expressions can be used by:** Regular Expressions are used by *grep* (and can be used) by *find* and many other programs.

**Tip:** If your regular expressions don't seem to be working then you probably need to use single quotation marks over the sentence and then use backslashes on every single special character.

. (dot)

will match *any single character*, equivalent to ? (question mark) in standard wildcard expressions. Thus, "m.a" matches "mpa" and "mea" but not "ma" or "mppa".

\ (backslash)

is used as an "escape" character, i.e. to protect a subsequent special character. Thus, "\\" searches for a backslash. Note you may need to use quotation marks and backslash(es).

.* (dot and asterisk)

is used to match any string, equivalent to * in standard wildcards.

* (asterisk)

the proceeding item is to be matched *zero or more* times. ie. n* will match n, nn, nnnn, nnnnnnn but not na or any other character.

^ (caret)

means "the beginning of the line". So "^a" means find a line starting with an "a".

$ (dollar sign)

means "the end of the line". So "a$" means find a line ending with an "a".

For example, this command searches the file myfile for lines starting with an "s" and ending with an "n", and prints them to the standard output (screen):

```
cat myfile | grep '^s.*n$'
```

[ ] (square brackets)

specifies a range. If you did m[a,o,u]m it can become: mam, mum, mom if you did: m[a-d]m it can become anything that starts and ends with m and has any character a to d inbetween. For example, these would work: mam, mbm, mcm, mdm. This kind of wildcard specifies an “or” relationship (you only need one to match).

|

This wildcard makes a logical OR relationship between wildcards. This way you can search for something or something else (possibly using two different regular expressions). You may need to add a '\' (backslash) before this command to work, because the shell may attempt to interpret this as a pipe.

[^]

This is the equivalent of [!] in standard wildcards. This performs a logical “not”. This will match anything that is not listed within those square brackets. For example, *rm myfile[^9]* will remove all myfiles* (ie. myfiles1, myfiles2 etc) but won't remove a file with the number 9 anywhere within it's name.

# Useful categories of characters (as defined by the POSIX standard)

This information has been taken from the grep info page with a tiny amount of editing, see [10] in the *Bibliography* for further information.

- [:upper:] uppercase letters
- [:lower:] lowercase letters
- [:alpha:] alphabetic (letters) meaning upper+lower (both uppercase and lowercase letters)
- [:digit:] numbers in decimal, 0 to 9
- [:alnum:] alphanumeric meaning alpha+digits (any uppercase or lowercase letters or any decimal digits)
- [:space:] whitespace meaning spaces, tabs, newlines and similar
- [:graph:] graphically printable characters excluding space
- [:print:] printable characters including space

- [:punct:] punctuation characters meaning graphical characters minus alpha and digits
- [:cntrl:] control characters meaning non-printable characters
- [:xdigit:] characters that are hexadecimal digits.

  **These are used with:** The above commands will work with most tools which work with text (for example: *tr*).

For example (advanced example), this command scans the output of the dir command, and prints lines containing a capital letter followed by a digit:

```
ls -l | grep '[[:upper:]][[:digit:]]'
```

The command greps for [upper_case_letter][any_digit], meaning any uppercase letter followed by any digit. If you remove the [ ] (square brackets) in the middle it would look for an uppercase letter or a digit, because it would become [upper_case_letter any_digit]

---

Prev Duplicating disks | Home | Up | Next Appendix