

**NAME**

File::Basename - Parse file paths into directory, filename and suffix.

**SYNOPSIS**

```
use File::Basename;

($name,$path,$suffix) = fileparse($fullname,@suffixlist);
$name = fileparse($fullname,@suffixlist);

$basename = basename($fullname,@suffixlist);
$dirname  = dirname($fullname);
```

**DESCRIPTION**

These routines allow you to parse file paths into their directory, filename and suffix.

**NOTE:** `dirname()` and `basename()` emulate the behaviours, and quirks, of the shell and C functions of the same name. See each function's documentation for details. If your concern is just parsing paths it is safer to use *File::Spec*'s `splitpath()` and `splitdir()` methods.

It is guaranteed that

```
# Where $path_separator is / for Unix, \ for Windows, etc...
dirname($path) . $path_separator . basename($path);
```

is equivalent to the original path for all systems but VMS.

```
fileparse
    my($filename, $directories, $suffix) = fileparse($path);
    my($filename, $directories, $suffix) = fileparse($path,
@suffixes);
    my $filename                = fileparse($path,
@suffixes);
```

The `fileparse()` routine divides a file path into its `$directories`, `$filename` and (optionally) the filename `$suffix`.

`$directories` contains everything up to and including the last directory separator in the `$path` including the volume (if applicable). The remainder of the `$path` is the `$filename`.

```
# On Unix returns ("baz", "/foo/bar/", "")
fileparse("/foo/bar/baz");

# On Windows returns ("baz", "C:\foo\bar\", "")
fileparse("C:\foo\bar\baz");

# On Unix returns ("", "/foo/bar/baz/", "")
fileparse("/foo/bar/baz/");
```

If `@suffixes` are given each element is a pattern (either a string or a `qr//`) matched against the end of the `$filename`. The matching portion is removed and becomes the `$suffix`.

```
# On Unix returns ("baz", "/foo/bar/", ".txt")
fileparse("/foo/bar/baz.txt", qr/\.[^.]*/);
```

If `type` is non-Unix (see `fileparse_set_fstype()`) then the pattern matching for suffix removal is performed case-insensitively, since those systems are not case-sensitive when opening existing files.

You are guaranteed that `$directories` . `$filename` . `$suffix` will denote the same location as the original `$path`.

`basename`

```
my $filename = basename($path);
my $filename = basename($path, @suffixes);
```

This function is provided for compatibility with the Unix shell command `basename(1)`. It does **NOT** always return the file name portion of a path as you might expect. To be safe, if you want the file name portion of a path use `fileparse()`.

`basename()` returns the last level of a filepath even if the last level is clearly directory. In effect, it is acting like `pop()` for paths. This differs from `fileparse()`'s behaviour.

```
# Both return "bar"
basename("/foo/bar");
basename("/foo/bar/");
```

`@suffixes` work as in `fileparse()` except all regex metacharacters are quoted.

```
# These two function calls are equivalent.
my $filename = basename("/foo/bar/baz.txt", ".txt");
my $filename = fileparse("/foo/bar/baz.txt", qr/\Q.txt\E/);
```

Also note that in order to be compatible with the shell command, `basename()` does not strip off a suffix if it is identical to the remaining characters in the filename.

`dirname`

This function is provided for compatibility with the Unix shell command `dirname(1)` and has inherited some of its quirks. In spite of its name it does **NOT** always return the directory name as you might expect. To be safe, if you want the directory name of a path use `fileparse()`.

Only on VMS (where there is no ambiguity between the file and directory portions of a path) and AmigaOS (possibly due to an implementation quirk in this module) does `dirname()` work like `fileparse($path)`, returning just the `$directories`.

```
# On VMS and AmigaOS
my $directories = dirname($path);
```

When using Unix or MSDOS syntax this emulates the `dirname(1)` shell function which is subtly different from how `fileparse()` works. It returns all but the last level of a file path even if the last level is clearly a directory. In effect, it is not returning the directory portion but simply the path one level up acting like `chop()` for file paths.

Also unlike `fileparse()`, `dirname()` does not include a trailing slash on its returned path.

```
# returns /foo/bar. fileparse() would return /foo/bar/
dirname("/foo/bar/baz");

# also returns /foo/bar despite the fact that baz is clearly a
# directory. fileparse() would return /foo/bar/baz/
dirname("/foo/bar/baz/");

# returns '.'. fileparse() would return 'foo/'
dirname("foo/");
```

Under VMS, if there is no directory information in the `$path`, then the current default device and directory is used.

`fileparse_set_fstype`

```
my $type = fileparse_set_fstype();
```

```
my $previous_type = fileparse_set_fstype($type);
```

Normally File::Basename will assume a file path type native to your current operating system (ie. /foo/bar style on Unix, \foo\bar on Windows, etc...). With this function you can override that assumption.

Valid \$types are "MacOS", "VMS", "AmigaOS", "OS2", "RISCOS", "MSWin32", "DOS" (also "MSDOS" for backwards bug compatibility), "Epc" and "Unix" (all case-insensitive). If an unrecognized \$type is given "Unix" will be assumed.

If you've selected VMS syntax, and the file specification you pass to one of these routines contains a "/", they assume you are using Unix emulation and apply the Unix syntax rules instead, for that function call only.

## SEE ALSO

*dirname(1)*, *basename(1)*, *File::Spec*