NAME

Digest::MD5 - Perl interface to the MD5 Algorithm

SYNOPSIS

```
# Functional style
use Digest::MD5 qw(md5 md5_hex md5_base64);

$digest = md5($data);
$digest = md5_hex($data);
$digest = md5_base64($data);

# OO style
use Digest::MD5;

$ctx = Digest::MD5->new;

$ctx->add($data);
$ctx->addfile($file_handle);

$digest = $ctx->digest;
$digest = $ctx->hexdigest;
$digest = $ctx->b64digest;
```

DESCRIPTION

The Digest::MD5 module allows you to use the RSA Data Security Inc. MD5 Message Digest algorithm from within Perl programs. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input.

Note that the MD5 algorithm is not as strong as it used to be. It has since 2005 been easy to generate different messages that produce the same MD5 digest. It still seems hard to generate messages that produce a given digest, but it is probably wise to move to stronger algorithms for applications that depend on the digest to uniquely identify a message.

The Digest::MD5 module provide a procedural interface for simple use, as well as an object oriented interface that can handle messages of arbitrary length and which can read files directly.

FUNCTIONS

The following functions are provided by the Digest::MD5 module. None of these functions are exported by default.

md5($data,...)

This function will concatenate all arguments, calculate the MD5 digest of this "message", and return it in binary form. The returned string will be 16 bytes long.

The result of md5("a", "b", "c") will be exactly the same as the result of md5("abc").

md5_hex($data,...)

Same as *md5()*, but will return the digest in hexadecimal form. The length of the returned string will be 32 and it will only contain characters from this set: '0'..'9' and 'a'..'f'.

md5_base64($data,...)

Same as *md5()*, but will return the digest as a base64 encoded string. The length of the returned string will be 22 and it will only contain characters from this set: 'A'..'Z', 'a'..'z', '0'..'9', '+' and '/'.

Note that the base64 encoded string returned is not padded to be a multiple of 4 bytes long. If you want interoperability with other base64 encoded md5 digests you might want to append the redundant string "==" to the result.

METHODS

The object oriented interface to `Digest::MD5` is described in this section. After a `Digest::MD5` object has been created, you will add data to it and finally ask for the digest in a suitable format. A single object can be used to calculate multiple digests.

The following methods are provided:

`$md5` = Digest::MD5->new

The constructor returns a new `Digest::MD5` object which encapsulate the state of the MD5 message-digest algorithm.

If called as an instance method (i.e. `$md5`->new) it will just reset the state the object to the state of a newly created object. No new object is created in this case.

`$md5`->reset

This is just an alias for `$md5`->new.

`$md5`->clone

This a copy of the `$md5` object. It is useful when you do not want to destroy the digests state, but need an intermediate value of the digest, e.g. when calculating digests iteratively on a continuous data stream. Example:

```
my $md5 = Digest::MD5->new;
while (<>) {
$md5->add($_);
print "Line $.: ", $md5->clone->hexdigest, "\n";
}
```

`$md5`->add($data,...)

The `$data` provided as argument are appended to the message we calculate the digest for. The return value is the `$md5` object itself.

All these lines will have the same effect on the state of the `$md5` object:

```
$md5->add("a"); $md5->add("b"); $md5->add("c");
$md5->add("a")->add("b")->add("c");
$md5->add("a", "b", "c");
$md5->add("abc");
```

`$md5`->addfile($io_handle)

The `$io_handle` will be read until EOF and its content appended to the message we calculate the digest for. The return value is the `$md5` object itself.

The *addfile()* method will *croak()* if it fails reading data for some reason. If it croaks it is unpredictable what the state of the `$md5` object will be in. The *addfile()* method might have been able to read the file partially before it failed. It is probably wise to discard or reset the `$md5` object if this occurs.

In most cases you want to make sure that the `$io_handle` is in `binmode` before you pass it as argument to the *addfile()* method.

`$md5`->add_bits($data, `$nbits`)
`$md5`->add_bits($bitstring)

Since the MD5 algorithm is byte oriented you might only add bits as multiples of 8, so you probably want to just use *add()* instead. The *add_bits()* method is provided for compatibility with other digest implementations. See Digest for description of the arguments that *add_bits()* take.

`$md5`->digest

Return the binary digest for the message. The returned string will be 16 bytes long.

Note that the `digest` operation is effectively a destructive, read-once operation. Once it has been performed, the `Digest::MD5` object is automatically `reset` and can be used to calculate

another digest value. Call `$md5->clone->digest` if you want to calculate the digest without resetting the digest state.

`$md5->hexdigest`
Same as `$md5->digest`, but will return the digest in hexadecimal form. The length of the returned string will be 32 and it will only contain characters from this set: '0'..'9' and 'a'..'f'.

`$md5->b64digest`
Same as `$md5->digest`, but will return the digest as a base64 encoded string. The length of the returned string will be 22 and it will only contain characters from this set: 'A'..'Z', 'a'..'z', '0'..'9', '+' and '/'.

The base64 encoded string returned is not padded to be a multiple of 4 bytes long. If you want interoperability with other base64 encoded md5 digests you might want to append the string "==" to the result.

EXAMPLES

The simplest way to use this library is to import the *md5_hex()* function (or one of its cousins):

```
use Digest::MD5 qw(md5_hex);
print "Digest is ", md5_hex("foobarbaz"), "\n";
```

The above example would print out the message:

```
Digest is 6df23dc03f9b54cc38a0fc1483df6e21
```

The same checksum can also be calculated in OO style:

```
use Digest::MD5;

$md5 = Digest::MD5->new;
$md5->add('foo', 'bar');
$md5->add('baz');
$digest = $md5->hexdigest;

print "Digest is $digest\n";
```

With OO style, you can break the message arbitrarily. This means that we are no longer limited to have space for the whole message in memory, i.e. we can handle messages of any size.

This is useful when calculating checksum for files:

```
use Digest::MD5;

my $filename = shift || "/etc/passwd";
open (my $fh, '<', $filename) or die "Can't open '$filename': $!";
binmode($fh);

$md5 = Digest::MD5->new;
while (<$fh>) {
$md5->add($_);
}
close($fh);
print $md5->b64digest, " $filename\n";
```

Or we can use the addfile method for more efficient reading of the file:

```
use Digest::MD5;

my $filename = shift || "/etc/passwd";
open (my $fh, '<', $filename) or die "Can't open '$filename': $!";
binmode ($fh);
```

```
print Digest::MD5->new->addfile($fh)->hexdigest, " $filename\n";
```

Since the MD5 algorithm is only defined for strings of bytes, it can not be used on strings that contains chars with ordinal number above 255 (Unicode strings). The MD5 functions and methods will croak if you try to feed them such input data:

```
use Digest::MD5 qw(md5_hex);

my $str = "abc\x{300}";
print md5_hex($str), "\n"; # croaks
# Wide character in subroutine entry
```

What you can do is calculate the MD5 checksum of the UTF-8 representation of such strings. This is achieved by filtering the string through *encode_utf8()* function:

```
use Digest::MD5 qw(md5_hex);
use Encode qw(encode_utf8);

my $str = "abc\x{300}";
print md5_hex(encode_utf8($str)), "\n";
# 8c2d46911f3f5a326455f0ed7a8ed3b3
```

SEE ALSO

Digest, Digest::MD2, Digest::SHA, Digest::HMAC

md5sum(1)

RFC 1321

http://en.wikipedia.org/wiki/MD5

The paper "How to Break MD5 and Other Hash Functions" by Xiaoyun Wang and Hongbo Yu.

COPYRIGHT

AUTHORS

The original `MD5` interface was written by Neil Winton (`N.Winton@axion.bt.co.uk`).

The `Digest::MD5` module is written by Gisle Aas <gisle@ActiveState.com>.