I saw Matz's Euroko 2010 keynote where he mentioned adding the ability to redefine methods gracefully in Ruby 2.0. I mean, more gracefully than this:

```ruby
class Hash
  # save the original method
  alias original_initialize initialize

  # redefine the method
  def initialize(*args, &block)
    # do something extra ...

    # call the original method
    __original_initialize__(*args, &block)

    # do something more ...
  end
end
```

I suggest adding a "redef" keyword to Ruby 2.0 for this purpose:

```ruby
class Hash
  # redefine the method
  redef initialize(*args, &block)
  # do something extra ...

  # call the original method
  super

  # do something more ...
end
```

The "redef" keyword should also have a twin "redefine_method" method for use in block-based metaprogramming:

```ruby
class Hash
  # redefine the method
  redefine_method :initialize do |*args, &block|
  # do something extra ...

    # call the original method
    super

    # do something more ...
  end
end
```
Thanks for your consideration.

end

History

#1 - 12/23/2010 03:24 PM - wardrop (Tom Wardrop)

begin
I came on here to suggest the exact same thing. As someone relatively new to Ruby (got into about 5 months ago), I'm surprised there's no easy way of achieving this given that so many developers seem to do this. It's what makes Ruby's dynamic nature so powerful.

There are numerous ways in which this could be implemented. You could introduce a new special method like super, called 'old' which could either hold a reference to the old implementation, or maybe instead, it might just return the old implementation as a Proc object?

I wouldn't want to see the 'super' method used for this purpose, as it may cause unpredictable behaviour. For example, what happens when you want to call the super class (the next implementation in the inheritance chain), but unexpectedly end up calling the old implementation which you'd thought you'd completely over-written.

I also wouldn't want to see anything like redef which can clutter up classes. There just needs to be some simple, clean way of getting at an overwritten method. A common use case is monkey patching (such a feature as this one would go well with the 'refinements' idea), where you want to prepend or append an operation to an existing method, or where you want to modify the return value.

I trust the Ruby developers and decision makers to come up with the most elegant and appropriate solution.
end

#2 - 12/23/2010 03:28 PM - wardrop (Tom Wardrop)

begin
Just to clarify, I'm not completely against the introduction of 'redef', it's just it would have to serve more a purpose than just allow access to the old implementation of a method. For example, if it became a requirement of the language to use redef when overwriting a non-inherited method, then obviously the introduction of 'redef' is a lot more justifiable.
end

#3 - 07/13/2011 12:47 AM - lazaridis.com (Lazaridis Ilias)

begin
See related issue #5005 with a suggested implementation of "original"
end

#4 - 10/18/2011 09:16 AM - naruse (Yui NARUSE)
- Project changed from Ruby master to 14
- Category deleted (core)
- Target version deleted (3.0)

#5 - 10/23/2011 05:21 PM - naruse (Yui NARUSE)
- Project changed from 14 to Ruby master

#6 - 03/18/2012 05:57 PM - shyouhei (Shyouhei Urabe)
- Description updated
- Status changed from Open to Assigned
- Assignee set to matz (Yukihiro Matsumoto)

#7 - 11/20/2012 09:15 PM - mame (Yusuke Endoh)
- Target version set to 3.0

#8 - 11/20/2012 11:07 PM - matz (Yukihiro Matsumoto)
- Status changed from Assigned to Rejected

To eliminate alias_method_chain, we introduced Module#prepend.
There's no chance to add redundant feature in the language.

Matz.