```
use subs 'chdir', 'chroot', 'chmod', 'chown';
chdir $somewhere;
sub chdir { ... }
```

To unambiguously refer to the built-in form, precede the built-in name with the special package qualifier `CORE::`. For example, saying `CORE::open()` always refers to the built-in `open()`, even if the current package has imported some other subroutine called `&open()` from elsewhere. Even though it looks like a regular function call, it isn't: you can't take a reference to it, such as the incorrect `\&CORE::open` might appear to produce.

Library modules should not in general export built-in names like `open` or `chdir` as part of their default `@EXPORT` list, because these may sneak into someone else's namespace and change the semantics unexpectedly. Instead, if the module adds that name to `@EXPORT_OK`, then it's possible for a user to import the name explicitly, but not implicitly. That is, they could say

```
use Module 'open';
```

and it would import the `open` override. But if they said

```
use Module;
```

they would get the default imports without overrides.

The foregoing mechanism for overriding built-in is restricted, quite deliberately, to the package that requests the import. There is a second method that is sometimes applicable when you wish to override a built-in everywhere, without regard to namespace boundaries. This is achieved by importing a sub into the special namespace `CORE::GLOBAL::`. Here is an example that quite brazenly replaces the `glob` operator with something that understands regular expressions.

```
package REGlob;
require Exporter;
@ISA = 'Exporter';
@EXPORT_OK = 'glob';

sub import {
    my $pkg = shift;
    return unless @_;
    my $sym = shift;
    my $where = ($sym =~ s/^GLOBAL_// ? 'CORE::GLOBAL' : caller(0));
    $pkg->export($where, $sym, @_);
}

sub glob {
    my $pat = shift;
    my @got;
    local *D;
    if (opendir D, '.') {
        @got = grep /$pat/, readdir D;
        closedir D;
    }
    return @got;
}
1;
```

And here's how it could be (ab)used:

```
#use REGlob 'GLOBAL_glob';      # override glob() in ALL namespaces
package Foo;
use REGlob 'glob';              # override glob() in Foo:: only
print for <^[a-z_]+\.pm\$>;     # show all pragmatic modules
```