



**POD Translation**  
by *pod2pdf*

---

[ajf@afco.demon.co.uk](mailto:ajf@afco.demon.co.uk)

*List-Compare*



# Table of Contents

## List-Compare

List::Compare:	1
Determining Relationships among Lists with Perl	1
YAPC::NA::2004	1
State University of New York at Buffalo	1
Friday, June 18, 2004, 9:50 am	1
James E. Keenan	1
Necessity Is the Mother of Invention of Perl Modules	1
Seen-Hashes as Lookup Tables	1
Repeated Code Is a Mistake	1
Why Not a Module to Get Information from Lists?	1
I Want It Faster!	2
I Want to Compare More Than 2 Lists!	2
3 or More Lists Are Trickier	2
What If I Want References, Not Lists?	2
Are These Items Found in Those Lists?	3
Are These Items Found in Any Lists?	3
What If I Already Have Seen-Hashes?	3
Can I Get It to Go Faster?	4
Why Bother with Objects?	4
Is the Interface Too Messy?	4
An Alternative Interface	5
My Hubris Leads to Your Laziness	5



## List::Compare:

### Determining Relationships among Lists with Perl

- YAPC::NA::2004
- State University of New York at Buffalo
- Friday, June 18, 2004, 9:50 am
- James E. Keenan
- To follow slides, go to:  
[http://mysite.verizon.net/jkeen/perl/YAPC/YAPC-NA-2004/List-Compare/slides/slide\\_001.html](http://mysite.verizon.net/jkeen/perl/YAPC/YAPC-NA-2004/List-Compare/slides/slide_001.html)

### Necessity Is the Mother of Invention of Perl Modules

- While preparing to teach Perl, had to keep track of 2 types of text files:
  - Plain-text source files for an HTML-based slideshow.
  - Perl demonstration scripts.
- Used *master* list to control order within each.
- Challenge: Was every file listed in the *master* list actually present in directory?

### Seen-Hashes as Lookup Tables

- Following *Perl Cookbook*, created 'seen-hashes' for each list
 

```
# loop through master file to populate @master
for (@master) { $seen_master{$_} = 1; }

# read directory holding source files to populate @sources
for (@sources) { $seen_sources{$_} = 1; }
```
- Then, ask whether master list is subset of source file list.
 

```
$subset_status = 1;
for (@master) {
    unless (exists $seen_sources{$_}) {
        $subset_status = 0;
        last;
    }
}
```
- Wrote similar code to keep track of demonstration Perl scripts.
- Soon got tired of repeating code for seen-hashes and subsets.

### Repeated Code Is a Mistake

- What I learned from Mark Jason Dominus:
  - Code repeated within a single script: Refactor into subroutine.
  - Code repeated across scripts: Refactor into module.

### Why Not a Module to Get Information from Lists?

- With a module, I could get a cleaner interface:
 

```
use List::Compare;
$lc = List::Compare->new(\@master, \@sources);
```

```
$subset_status = $lc->is_LsubsetR();
```

- But why stop at just subset relationships? How about these:
 

```
@intersection = $lc->get_intersection();
@union        = $lc->get_union();
@unique       = $lc->get_unique();
@complement   = $lc->get_complement();
@symmetric_difference = $lc->get_symmetric_difference;
```

*Perl Cookbook* uses seen-hashes to derive these relationships between 2 lists.

- Brainstorm: If I modularized this code, I'd never have to re-type it in a script.

### I Want It Faster!

- List::Compare's Regular mode computes all relationships inside the constructor.
- Challenge: Why have constructor compute all relationships if you only want one?
- Response: List::Compare's Accelerated Mode.
 

```
$lca = List::Compare->new('-a', \@master, \@sources);
```

```
@intersection = $lca->get_intersection;
```

### I Want to Compare More Than 2 Lists!

- Challenge: Why should I be limited to comparing only 2 lists at a time?
 

```
@Al      = qw(abel abel baker camera delta edward fargo golfer);
@Bob     = qw(baker camera delta delta edward fargo golfer hilton);
@Carmen  = qw(fargo golfer hilton icon icon jerky kappa);
@Don     = qw(fargo icon jerky);
@Ed      = qw(fargo icon icon jerky);
```
- Response: List::Compare's Multiple mode ... which looks just like the Regular mode.
 

```
$lcm = List::Compare->new(\@Al, \@Bob, \@Carmen, \@Don, \@Ed);
```

```
@intersection = $lcm->get_intersection;
```

### 3 or More Lists Are Trickier

Challenge: How would I get items unique to @Carmen?

```
@Al      = qw(abel abel baker camera delta edward fargo golfer);
@Bob     = qw(baker camera delta delta edward fargo golfer hilton);
@Carmen  = qw(fargo golfer hilton icon icon jerky kappa);
@Don     = qw(fargo icon jerky);
@Ed      = qw(fargo icon icon jerky);
```

Response: Pass @Carmen's index position in constructor's @\_ as argument to get\_unique().

```
$lcm = List::Compare->new(\@Al, \@Bob, \@Carmen, \@Don, \@Ed);
#           0       1       2       3       4
```

```
@unique_Carmen = $lcm->get_unique(2);
```

### What If I Want References, Not Lists?

- Challenge: Most List::Compare methods return a list. What if I only need that list as input to some other function?
 

```
@union = $lc->get_union;
some_other_function(@union);
```

Wouldn't it be faster if I just returned and passed an array reference?

-

Response: Parallel methods which return references

```
$unionref = $lcm->get_union_ref;
some_other_function($unionref);
```

### Are These Items Found in Those Lists?

Challenge: Sometimes we want to know *in which* of several lists one or more items can be found.

Response: Two new methods: `is_member_which()` and `are_members_which()`.

- ```
@memb_arr = $lcm->is_member_which('golfer');

# @memb_arr will hold: ( 0, 1, 2 )
```
- ```
$memb_hash_ref = $lcm->are_members_which(
    [ qw| abel baker fargo hilton zebra | ] );

# $memb_hash_ref will be:

{
    abel    => [ 0
              ],
    baker   => [ 0, 1
              ],
    fargo   => [ 0, 1, 2, 3, 4
              ],
    hilton  => [ 1, 2
              ],
    zebra   => [
              ],
};
```

### Are These Items Found in Any Lists?

- Challenge: Sometimes we want to know whether one or more items were found *in any* of several lists
- Response: Two new methods which return Boolean(-ish) values

- ```
$found = $lcm->is_member_any('abel');
# $found will be: 1
```
- ```
$memb_hash_ref = $lcm->are_members_any(
    [ qw| abel baker fargo hilton zebra | ] );

# $memb_hash_ref will be:

{
    abel    => 1,
    baker   => 1,
    fargo   => 1,
    hilton  => 1,
    zebra   => 0,
};
```

### What If I Already Have Seen-Hashes?

- Challenge: Sometimes we've already computed seen-hashes.
 

```
%seenAl = ( abel    => 2, baker    => 1, camera => 1,
            delta   => 1, edward  => 1, fargo  => 1,
            golfer  => 1
            );

%seenBob = (
            baker    => 1, camera => 1,
            delta   => 2, edward  => 1, fargo  => 1,
            golfer  => 1, hilton  => 1
            );
```

Since List::Compare *internally* transforms lists into seen-hashes, why can't we just pass the seen-hashes directly?

- Response: Now we can.
 

```
$lcsH = List::Compare->new(\%seenAl, \%seenBob);

@intersection = $lcsH->get_intersection;
```

### Can I Get It to Go Faster?

- By default, List::Compare sorts the lists its methods returns.
- You can get a small speed boost if you pass the Unsorted option to the constructor.
 

```
$lcu = List::Compare->new('-u', \@Llist, \@Rlist);

or

$lcu = List::Compare->new('--unsorted', \@Llist, \@Rlist);

@intersection = $lcu->get_intersection;
# @intersection will not be sorted
```

### Why Bother with Objects?

- Challenge: Why bother with the overhead cost of creating a List::Compare object?
- Response: A faster but less elegant interface: List::Compare::Functional
 

```
use List::Compare::Functional qw( get_union get_complement );

@union = get_union( [ \@Al, \@Bob, \@Carmen, \@Don, \@Ed ] );
```
- No constructor, so lists must be passed each time a function is called.
- References to lists are themselves placed in a list. A reference to that is passed to the function.
- Where a function needs extra arguments, these must also be wrapped in an array which is passed by reference to the function.
 

```
@complement_Don =
  get_complement( [ \@Al, \@Bob, \@Carmen, \@Don, \@Ed ],
                 [ 3 ] );
```

### Is the Interface Too Messy?

- Challenge: Some might say that List::Compare::Functional's interface is not very self-documenting.
 

```
use List::Compare::Functional qw(
  get_complement
  is_LsubsetR
  are_members_which
);

@complement = get_complement(
  '-u',
  [ \@Al, \@Bob, \@Carmen, \@Don, \@Ed ],
  [3],
);

$LR = is_LsubsetR(
  [ \@Al, \@Bob, \@Carmen, \@Don, \@Ed ],
  [2,3],
);

$memb_hash_ref = are_members_which(
  [ \@Al, \@Bob, \@Carmen, \@Don, \@Ed ],
  [ qw| abel baker fargo hilton zebra | ],
);
```



- You have to get the order of the arguments just right — and can you tell what each array reference means?

### An Alternative Interface

- Response: David H. Adler suggested passing a single hash reference with named arguments:
 

```
@complement = get_complement( {
    lists    => [ \@Al, \@Bob, \@Carmen, \@Don, \@Ed ],
    item     => 3,
    unsorted => 1,
} );

$LR = is_LsubsetR( {
    lists    => [ \@Al, \@Bob, \@Carmen, \@Don, \@Ed ],
    pair     => [ 2, 3 ],
} );

$memb_hash_ref = are_members_which( {
    lists    => [ \@Al, \@Bob, \@Carmen, \@Don, \@Ed ],
    items    => [ qw| abel baker fargo hilton zebra | ],
} );
```
- More verbose, but more self-documenting.
- The order in which arguments are passed no longer matters, but you have to get the names of the keys right.
- Now available for both List::Compare and List::Compare::Functional. See documentation for version 0.29 or later.

### My Hubris Leads to Your Laziness

- To compare lists, you never have to code up a seen-hash again.
- Just use `List::Compare`;
- Get it: <http://search.cpan.org/~jkeenan/List-Compare-0.30/> or <http://mysite.verizon.net/jkeen/perl/modules/List-Compare/>
- Kudos and complaints: [jkeenan@cpan.org](mailto:jkeenan@cpan.org)
- Inspirations:
  - *Perl Cookbook* (2nd ed.), Tom Christiansen and Nathan Torkington, O'Reilly & Associates, 2003.
  - Program Repair Shop and Red Flags, Mark Jason Dominus, <http://www.perl.com/lpt/a/2000/11/repair3.html>
- For further reading: *The Perl Journal*, May 2004, <http://www.tpj.com>

