

# Using the ecCodes GRIB Tools: comparison and modification tools

Computer User Training Course 2016

**Paul Dando**

User Support Section

[advisory@ecmwf.int](mailto:advisory@ecmwf.int)



© ECMWF 5 October 2016

## Contents

- Comparing messages
- Copying messages
- Setting key / value pairs

## `grib_compare` – compare GRIB messages

- Use `grib_compare` to compare the GRIB messages contained in two files
- By default, messages are compared in the same order, bit-by-bit and with floating point values compared exactly
  - Tolerances for data values can be specified based on the absolute, relative or packing error
  - Default tolerance is absolute error = 0
- If differences are found `grib_compare`
  - switches to a key-based mode to find out which coded keys are different
  - **fails** returning a non-zero exit code
- Options are available to compare only specific keys or sets of keys

## grib\_compare – basic usage

```
grib_compare [options] grib_file grib_file
```

- Options

<code>-b key1, key2, ...</code>	All keys in this list are skipped when comparing files
<code>-c key1, namespace2:n...</code>	Compare these keys only
<code>-H</code>	Compare message headers only
<code>-e</code>	Edition-independent compare
<code>-w key1=val1, key2!=val2, ...</code>	Where option
<code>-f</code>	Do <i>not</i> fail on error
<code>-r</code>	Messages not in the same order
<code>-v</code>	Verbose

## `grib_compare` – a simple example

- Two GRIB messages in `f1.grib1` and `f2.grib1` contain the land-sea mask at different forecast time steps

```
> grib_compare f1.grib1 f2.grib1
-- GRIB #1 -- shortName=lsn paramId=172 stepRange=3
  levelType=sfc level=0 packingType=grid_simple
  gridType=reduced_gg --
long [P1]: [3] != [6]

> echo $?
1
```

- The exit code is set to 1 because the comparison failed

## `grib_compare` – a simple example

- If we blacklist the key `P1` and compare the files again

```
> grib_compare -b P1 f1.grib1 f2.grib1  
  
> echo $?  
0
```

- The exit code is set to 0 because the comparison is successful according to the blacklist

## `grib_compare` – verbose output

- The verbose option shows details of all keys being compared

```
> grib_compare -v f1.grib1 f2.grib1
f1.grib1
  comparing totalLength as long
  comparing editionNumber as long
  comparing sectionLength as long
  comparing table2Version as long
  comparing centre as string
  comparing generatingProcessIdentifier as long
  comparing gridDefinition as long
  ...
  comparing P1 as long
-- GRIB #1 -- shortName=lsn paramId=172 stepRange=3 levelType=sfc
evel=0 packingType=grid_simple gridType=reduced_gg --
long [P1]: [3] != [6]
  comparing P2 as long
  ...
```

## `grib_compare` – limit the keys compared

- The `-c` option can be used to compare only specific keys

```
> grib_compare -c dataDate f1.grib1 f2.grib1
-- GRIB #1 -- shortName=lsn paramId=172 stepRange=3
  levelType=sfc level=0 packingType=grid_simple
  gridType=reduced_gg --
long [dataDate]: [20160224] != [20160225]
```

- Or a set of keys in a particular namespace

```
> grib_compare -c time:n f1.grib1 f2.grib1
-- GRIB #1 -- shortName=lsn paramId=172 stepRange=3
  levelType=sfc level=0 packingType=grid_simple
  gridType=reduced_gg --
long [dataDate]: [20160224] != [20160225]
long [validityDate]: [20160224] != [20160225]
```



## `grib_compare` – compare headers only

- To compare only the headers of two GRIB messages use the `-H` option

```
> grib_compare -H f1.grib1 f2.grib1
-- GRIB #1 -- shortName=lsn paramId=172 stepRange=3
  levelType=sfc level=0 packingType= gridType= --
long [day]: [24] != [25]
```

- The `-H` option cannot be used with the `-c` option

## `grib_compare` – edition-independent

- Two GRIB messages are very different if they are encoded with different editions

```
> grib_compare sp.grib1 sp.grib2
-- GRIB #1 -- shortName=sp paramId=134 stepRange=0 levelType=sfc
   level=0 packingType=grid_simple gridType=reduced_gg --
long [totalLength]: [4284072] != [4284160]
long [editionNumber]: [1] != [2]
long [sectionLength]: [52] != [21]
[table2Version] not found in 2nd field
[gridDefinition] not found in 2nd field
[indicatorOfParameter] not found in 2nd field
[indicatorOfTypeOfLevel] not found in 2nd field
[yearOfCentury] not found in 2nd field
[unitOfTimeRange] not found in 2nd field
[P1] not found in 2nd field
[P2] not found in 2nd field
...
```

## `grib_compare` – edition-independent

- Using the `-e` option `grib_compare` compares only the higher level information common to the two messages

```
> grib_compare -e sp.grib1 sp.grib2
-- GRIB #1 -- shortName=sp paramId=134 stepRange=0
  levelType=sfc level=0 packingType=grid_simple
  gridType=reduced_gg --
string [param]: [134.128] != [134]
```

- The two messages contain the same information encoded in two different ways
- Only the MARS param is different

## `grib_compare` – summary of differences

- When files contain several fields and some keys are different, it is useful to have a summary report

```
> grib_compare -f f1.grib1 f2.grib1
-- GRIB #1 -- shortName=z paramId=129 stepRange=0 levelType=pl
   level=1000 packingType=spectral_complex gridType=sh --
long [marsType]: [an] != [fc]

-- GRIB #3 -- shortName=z paramId=129 stepRange=0 levelType=pl
   level=850 packingType=spectral_complex gridType=sh --
long [marsType]: [an] != [fc]
...
## ERRORS SUMMARY #####
##
## Summary of different key values
## marsType ( 6 different )
##
## 6 different messages out of 12
```

## `grib_compare` – comparing data values

- By default floating point values are compared exactly
- Different tolerances can be provided using one of the following options

<code>-A absolute_error</code>	Use absolute error as tolerance
<code>-R key=rel_error,...</code>	Use relative error as tolerance for <code>key</code>
<code>-P</code>	Use packing error as tolerance
<code>-T factor</code>	Compare data values using tolerance specified in options <code>-A</code> , <code>-R</code> , <code>-P</code> multiplied by an integer <code>factor</code>

## `grib_compare` – setting the tolerance

- Comparison of the data values in two files shows that one of the seven values is different with the default absolute error tolerance of zero

```
> grib_compare -c data:n f1.grib1 f2.grib1
-- GRIB #1 -- shortName=2t paramId=167 stepRange=0 levelType=sfc level=0
  packingType=grid_simple gridType=reduced_gg --
double [packedValues]: 1 out of 7 different
max absolute diff. = 2.0000000000000000e+00, relative diff. = 0.4
  max diff. element 2: 3.0000000000000000e+00
  5.0000000000000000e+00
  tolerance=0.0000000000000000e+00 packingError: [0.0625005]
[0.0625005]
  values max= [70] [70]          min= [1] [1]
```

- Set the absolute error tolerance to 2.0 and the comparison is successful

```
> grib_compare -A 2.0 -c data:n f1.grib1 f2.grib1
```

## grib\_compare – setting the tolerance

- We can also set a relative error as tolerance for each key

```
> grib_compare -c data:n f1.grib1 f2.grib1
-- GRIB #1 -- shortName=2t paramId=167 stepRange=0 levelType=sfc level=0
  packingType=grid_simple gridType=reduced_gg --
double [packedValues]: 1 out of 7 different
  max absolute diff. = 2.0000000000000000e+00, relative diff. = 0.4
  max diff. element 2: 3.0000000000000000e+00
  5.0000000000000000e+00
  tolerance=0.0000000000000000e+00 packingError: [0.0625005]
  [0.0625005]
  values max= [70] [70] min= [1] [1]
values max= [70] [70] min= [1] [1]
```

- Set a relative error of 0.4 as the tolerance for packedValues

```
> grib_compare -R packedValues=0.4 -c data:n f1.grib1 f2.grib1
```

- The comparison is successful because the relative tolerance is greater than the relative difference

## grib\_compare – setting the tolerance

- Different packing precision can give different data values

```
> grib_compare -c data:n f1.grib1 f3.grib1
-- GRIB #1 -- shortName=2t paramId=167 stepRange=0 levelType=sfc level=0
  packingType=grid_simple gridType=reduced_gg --
double [packedValues]: 1 out of 7 different
max absolute diff. = 5.0000000000000000e-01, relative diff. = 0.166667
  max diff. element 1: 2.5000000000000000e+00
  3.0000000000000000e+00
  tolerance=0.0000000000000000e+00 packingError: [0.0625005] [0.5]
  values max= [70] [70] min= [1] [1]
values max= [70] [70] min= [1] [1]
```

- Here we can use the packing error as the tolerance

```
> grib_compare -P -c data:n f1.grib1 f3.grib1
```

- The comparison is successful because the maximum absolute difference is within the larger of the two packing errors – only the packing precision has changed



## Practicals

- Work in your \$SCRATCH

```
cd $SCRATCH
```

- Make a copy of the practicals directory in your \$SCRATCH

```
tar -xvf /home/etrain/trx/ecCodes/grib_tools2.tar
```

- There is a sub-directory for each practical:

```
ls $SCRATCH/grib_tools2
```

```
compare filter modify
```

## Practical: using grib\_compare

1. Use `grib_compare` to compare the GRIB messages contained in the files `file1.grib` and `file2.grib`
  - Which keys does `grib_compare` report as different ? What is the exit code returned ?
2. Now use the `-b` option to 'black list' the keys that you know are different and use `grib_compare` to compare the messages again
  - Are any keys reported as different ? What is the exit code ?
3. Compare the data namespaces (use "`grib_compare -c data:n ...`" ) for `file1.grib` and `file2.grib`. What values need to be set for the absolute (with `-A`) and relative (with `-R`) error tolerances for the comparison to be successful ?
4. Repeat question 3 with `file2.grib` and `file3.grib` [More difficult]

## `grib_copy` – copy contents of GRIB files

- Use `grib_copy` to copy selected messages from GRIB files optionally printing some key values
- Without options `grib_copy` prints **no** key information
- Options exist to specify the set of keys to print
  - Use verbose option (`-v`) to print keys
- Output can be ordered
  - E.g. order by ascending or descending step
- Key values can be used to specify the output file names
- `grib_copy` **fails** if a key is not found
  - Use the `-f` option to force `grib_copy` not to fail on error

## `grib_copy` – usage

```
grib_copy [options] grib_file grib_file ... out_grib_file
```

- Options

`-p key1, key2, ...`

Keys to print (only with `-v`)

`-w key=val1, key2!=val2, ...`

Where option

`-B "key1 asc, key2 desc"`

Order by: "step asc, centre desc"

`-v`

Verbose

`-f`

Do *not* fail on error

...

## grib\_copy – examples

- To copy only the analysis fields from a file

```
> grib_copy -w dataType=an in.grib1 out.grib1
```

- To copy only those fields that are not analysis fields

```
> grib_copy -w dataType!=an in.grib1 out.grib1
```

- Information can be output using the `-v` and `-p` options

```
> grib_copy -v -p shortName in.grib1 out.grib1
in.grib1
shortName
t
1 of 1 grib messages in in.grib1
1 of 1 total grib messages in 1 files
```

## grib\_copy – using key values in output file

- Key values can be used to specify the output file name

```
> grib_copy in.grib "out_[shortName].grib"  
> ls out_*  
out_2t.grib  out_ms1.grib ...
```

*Use quotes to protect the [ ]s*

- This provides a convenient way to filter GRIB messages into separate files

## `grib_set` – set key / value pairs

- Use `grib_set` to
  - Set key / value pairs in the input GRIB file(s)
  - Make simple changes to key / value pairs in the input GRIB file(s)
- Each GRIB message is written to the output file
  - By default this includes messages for which no keys are changed
  - With `-s` (strict) option **only** messages matching **all constraints** in the `where` option are copied
- An option exists to repack data
  - Sometimes after setting some keys involving properties of the packing algorithm the data needs to be repacked
- `grib_set` **fails** when an error occurs
  - e.g. when a key is not found

## `grib_set` – usage

```
grib_set [options] grib_file grib_file ... out_grib_file
```

- Options

<code>-s key1=val1,key2=val2,...</code>	List of key / values to set
<code>-p key1,key2,,...</code>	Keys to print (only with <code>-v</code> )
<code>-w key1=val1,key2!=val2,...</code>	Where option
<code>-d value</code>	Set all data values to <code>value</code>
<code>-f</code>	Do <i>not</i> fail on error
<code>-v</code>	Verbose
<code>-S</code>	Strict
<code>-r</code>	Repack data
<code>...</code>	



## `grib_set` – examples

- To set the parameter value of a field to 10m wind speed (10si)

```
> grib_set -s shortName=10si in.grib1 out.grib1
```

- This changes e.g.
  - `shortName` to 10si
  - `paramId` to 207
  - `name` / `parameterName` to '10 metre wind speed'
  - `units` / `parameterUnits` to 'm s \*\* -1'
  
  - `indicatorOfParameter` to 207 **GRIB edition dependent !**
  - `marsParam` to 207.128

## grib\_set – examples

- Some keys are read-only and cannot be changed directly

```
> grib_set -s name="10 metre wind speed" in.grib1
out.grib1

ECCODES ERROR   : grib_set_values[0] name (3)
                  failed: Value is read only
```

- The read-only keys can be set only by setting one of the other keys, e.g.
  - `shortName=10si`
  - `paramId=207`
  - `indicatorOfParameter=207`      **GRIB edition dependent !**

## `grib_set` – set key values to missing

- When a key is not used all the bits of its value should be set to 1 to indicate that it is 'missing'
- Different keys have different lengths so the value that needs to be coded for missing keys is not unique
- To set a key to missing a string "missing" or "MISSING" is accepted by `grib_set`

```
> grib_set -s Ni=missing in.grib2 out.grib2
```

- Note that some values cannot be set to "missing" !

```
> grib_set -s dataDate=missing file1.grib2 file2.grib2
ECCODES ERROR   :  unable to set dataDate=missing (Value cannot be missing)
ECCODES ERROR   :  grib_set_values[0] dataDate (7) failed: Value cannot be missing
```

## `grib_set` – changing decimal precision

- To pack a temperature expressed in Kelvin with 1 digit of precision after the decimal point we can set `changeDecimalPrecision=1`
  - N.B. this is different to setting the number of significant digits !

```
> grib_set -s changeDecimalPrecision=1 T.grib1 T1.grib1
```

- Use `grib_compare` to see the differences

```
> grib_compare -c data:n T.grib1 T1.grib1
-- GRIB #1 -- shortName=2t paramId=167 stepRange=0 levelType=sfc level=0
  packingType=grid_simple gridType=reduced_gg --
double [packedValues]: 2132215 out of 2140702 different
max absolute diff. = 5.0000000000011369e-02, relative diff. = 0.000207239
max diff. element 17: 2.41216796875000000000e+02 2.41266796875000011369e+02
tolerance=0.0000000000000000e+00 packingError: [0.000984192] [0.0500122]
values max= [315.447] [315.467] min= [216.967] [216.967]
```

## `grib_set` – modify data values

- An offset can be added to all data values in a GRIB message by setting the key `offsetValuesBy`

```
> grib_get -F "%.5f" -p max,min,average TK.grib  
315.44727 216.96680 286.34257  
  
> grib_set -s offsetValuesBy=-273.15 TK.grib TC.grib  
  
> grib_get -F "%.5f" -p max,min,average TC.grib  
42.29726 -56.18321 13.19257
```

## `grib_set` – modify data values

- The data values in a GRIB message can be multiplied by a factor by setting the key `scaleValuesBy`

```
> grib_get -F "%.2f" -p max,min,average Z.grib
65035.92 -3626.08 2286.30

> grib_set -s scaleValuesBy=0.102 Z.grib1 orog.grib1

> grib_get -F "%.2f" -p max,min,average orog.grib1
6633.64 -369.86 233.20
```

## `grib_set` – using key values in output file

- Key values can be used to specify the output file name

```
> grib_set -s time=0000 in.grib "out_[shortName].grib"  
  
> ls out_*  
out_2t.grib  out_ms1.grib ...
```

- Remember: Use quotes to protect the [ ]s !

## What **cannot** be done with `grib_set`

- `grib_set` cannot be used for making transformations to the data representation
  - It cannot be used to transform data from spectral to grid-point representation (and vice-versa)
- `grib_set` cannot be used transform data from one grid representation to another
  - It cannot be used to transform data from regular or reduced Gaussian grids to regular latitude-longitude grids
- `grib_set` cannot be used to select sub-areas of data
  - It will change the value of, e.g. `latitudeOfFirstGridPointInDegrees` etc, but the data will still be defined on the original grid
- **The GRIB tools cannot be used to interpolate the data**