

Using the ecCodes GRIB Tools

Computer User Training Course 2017

Paul Dando

User Support Section

advisory@ecmwf.int

Contents

- Getting key / value pairs
- Getting data values
- Comparing messages
- Copying messages
- Setting key / value pairs
- Converting from GRIB to NetCDF

`grib_get` – get key / value pairs

- Use `grib_get` to get the values of one or more keys from one or more GRIB files – very similar to `grib_ls`
- By default `grib_get` **fails** if an error occurs (e.g. key not found) returning a non-zero exit code
 - Suitable for use in scripts to obtain key values from GRIB messages
 - Can force `grib_get` not to fail on error
- Options available to get all MARS keys or all keys for a particular namespace
 - Can get other keys in addition to the default set
- Format of floating point values can be controlled with a C-style format statement
- `grib_get` can also be used to find the grid point(s) nearest to a specified latitude-longitude and print the value of the field at that point
 - Works in the same way as `grib_ls`

grib_get – usage

```
grib_get [options] grib_file grib_file ...
```

- Options

`-p key1,key2,...`

Keys to get

`-P key1,key2,...`

Additional keys to get with `-m`, `-n`

`-w key1=val1,key2!=val2,...`

Where option

`-s key1=val1,key2=val2,...`

Keys to set (temporary for printing)

`-m`

Get all MARS keys

`-n namespace`

Get all keys for `namespace`

`-l lat,lon[,MODE,FILE]`

Value(s) nearest to lat-lon point

`-F format`

Format for floating point values

`-f`

Do *not* fail on error

...

Using the 'where' option

- The where option `-w` can be used with all the GRIB Tools
- Constraints are of the form `key=value` or `key!=value` or `key=value1/value2/value2`

`-w key1=value1 ,key2 :i!=value2 ,key3 :s=value3`

- Messages are processed only if they match **ALL** the key / value constraints

```
> grib_get -w level=100 -p step file.grib1          "IS"
...
> grib_get -w level!=100 -p step file.grib1        "NOT"
...
> grib_get -w level=100,step=3 -p step file.grib1   "AND"
...
> grib_get -w level=100/200/300 -p step file.grib1 "OR"
...
```

Specifying the type of the key

- All ecCodes keys have a default type
 - e.g. string, integer, floating point
- The type of the key can be specified as follows:
 - **key** → native type
 - **key:i** → integer
 - **key:s** → string
 - **key:d** → double

```
> grib_get -p centre:i,dataDate,shortName,paramId,typeOfLevel,level file.grib1
98          20170228      t          130          isobaricInhPa 1000
```

`grib_get` – examples

- To get the centre of the first (`count=1`) GRIB message in a file (both as a ‘string’ and an ‘integer’)

```
> grib_get -w count=1 -p centre f1.grib1
ecmf

> grib_get -w count=1 -p centre:i f1.grib1
98
```

- `grib_get` fails if there is an error

```
> grib_get -p mykey f1.grib1
ECCODES ERROR      : Key/value not found
```

```
> echo $?
246
```

← returns the exit code from the previous command

grib_get – examples

- To get all the MARS keys, optionally printing the `shortName`

```
> grib_get -m f1.grib1
g sfc 20170228 1200 0 167.128 od an oper 0001

> grib_get -m -P shortName f1.grib1
2t g sfc 20170228 1200 0 167.128 od an oper 0001
```

- To get all keys belonging to the statistics namespace

```
> grib_get -n statistics f1.grib1
314.24 214.613 277.111 21.0494 41379.8 2.48314e-05 0
```

- `grib_get -m` is the same as `grib_get -n mars`

`grib_get` – controlling output format

- The format of floating point values can be controlled by using a C-style format statement with the `-F` option
 - F `"%.4f"` - Decimal format with 4 decimal places (1.2345)
 - F `"%.4e"` - Exponent format with 4 decimal places (1.2345E-03)

```
> grib_get -F "%.6f" -p maximum f1.grib1
```

```
314.240280
```

```
> grib_get -F "%.4e" -p maximum f1.grib1
```

```
3.1424e+02
```

- Default format is `-F "%.10e"`

`grib_get` – `stepRange` and `stepUnits`

- The step is always printed as an **integer** value
- By default the units of the step are printed in hours
- To obtain the step in other units set the **stepUnits** appropriately with the `-s` option

```
> grib_get -p stepRange f1.grib1
```

```
6
```

```
12
```

```
> grib_get -s stepUnits=m -p stepRange f1.grib1
```

```
360
```

```
720
```

stepUnits can be s, m, h, 3h, 6h, 12h, D, M, Y, 10Y, 30Y, C

Finding nearest grid points with grib_get

- The value of a GRIB field close to a specified point of Latitude/Longitude can be found with `grib_get`
 - Works in the same way as `grib_ls`

```
> grib_get -l 52.0,-1.43 f1.grib1  
  
273.58 272.375 273.17 273.531  
  
> grib_get -F "%.5f" -P stepRange -l 52.0,-1.43,1 f1.grib1  
  
0 272.37505
```

- GRIB files specified must contain grid point data

Getting data values at a grid point

- The value of a GRIB field at a particular grid point can be printed using `grib_get` with the `-i` option
- For example, find the index of a nearest grid point with `grib_ls` and then use this with `grib_get` to build a list of values at that point:

```
> grib_get -F "%.2f" -i 2159 -p step,dummy:s f1.grib1
```

```
6 99429.31
```

```
12 99360.25
```

```
18 99232.31
```

```
24 99325.56
```

*Forces a space
between step and
value*

- Also returns a value for non-grid point data !

`grib_get_data` – print data values

- Use `grib_get_data` to print a list of latitude, longitude (for grid point data) and data values from one or more GRIB files
- The format of the output can be controlled by using a C-style format statement with the `-F` option
 - F `"%.4f"` - Decimal format with 4 decimal places (1.2345)
 - F `"%.4e"` - Exponent format with 4 decimal places (1.2345E-03)The default format is `-F "%.10e"`
- By default missing values are not printed
 - A user-provided string can be printed in place of any missing values
- By default `grib_get_data` **fails** if there is an error
 - Use the `-f` option to force `grib_get_data` not to fail on error

grib_get_data – usage

```
grib_get_data [options] grib_file grib_file ...
```

Options

<code>-p key1,key2,...</code>	Keys to print
<code>-w key1=val1,key2!=val2,...</code>	Where option
<code>-m missingValue</code>	Specify missing value string
<code>-F format</code>	C-style format for output values
<code>-f</code>	Do <i>not</i> fail on error
<code>...</code>	

grib_get_data – example

```
> grib_get_data -F "%.4f" f1.grib1
```

Latitude,	Longitude,	Value
81.000	0.000	22.5957
81.000	1.500	22.9009
81.000	3.000	22.8359
81.000	4.500	22.3379
81.000	6.000	21.5547
81.000	7.500	20.7344
81.000	9.000	19.8916
81.000	10.500	18.5747
81.000	12.000	17.2578
81.000	13.500	16.1343
81.000	15.000	14.9785
81.000	16.500	13.8296

...

*Format option
applies to values
only - not to the
Latitudes and
Longitudes*

grib_get_data – missing values example

```
> grib_get_data -m XXXXX -F "%.4f" f1.grib1
```

```
Latitude, Longitude, Value
```

```
...
```

```
81.000 90.000 9.4189
```

```
81.000 91.500 8.6782
```

```
81.000 93.000 XXXXX
```

```
81.000 94.500 XXXXX
```

```
81.000 96.000 XXXXX
```

```
81.000 97.500 XXXXX
```

```
81.000 99.000 6.7627
```

```
81.000 100.500 7.4097
```

```
81.000 102.000 7.9307
```

```
...
```

*Missing values are
printed with
XXXXXX*

Practicals

- Work in your \$SCRATCH

```
cd $SCRATCH
```

- There is a sub-directory for each practical:

```
ls $SCRATCH/grib_tools
```

```
compare  grib_dump
```

```
filter   grib_get   grib_ls
```

```
modify
```

Reminder: If you need to get the material for the GRIB tools practicals:

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

```
tar -xvf /home/ectrain/trx/ecCodes/grib_tools.tar
```

- This will create a directory in your \$SCRATCH containing the GRIB data files for all the GRIB tools practicals

Practical: using grib_get & grib_get_data

1. Use `grib_get` to obtain a list of all the pressure levels available for parameter T in the file `tz_an_pl.grib1`
2. Use `grib_get` to print the `shortName`, `dataTime`, `dataDate` and `level` for the 500 & 1000 hPa levels only in `tz_an_pl.grib1`
3. Use `grib_get` to print the `stepRange` for the fields in the file `surface.grib1` in (a) hours (b) minutes and (c) seconds – what happens ?
4. Use `grib_get_data` to print the latitude, longitude and values for the first field in `surface.grib1`
 - Output results in decimal format with 5 decimal places
 - Output results in exponential format with 10 decimal places
 - Are there any missing values ?
5. Use `grib_get_data` to print the data values for the temperature at 500 hPa **only** from the file `tz_an_pl.grib1`
 - Make sure you print only the data for T500 ! What is printed ?

`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

`-b key1, key2, ...`

All keys in this list are skipped when comparing files

`-c key1, namespace2:n...`

Compare these keys only

`-H`

Compare message headers only

`-e`

Edition-independent compare

`-w key1=val1, key2!=val2, ...`

Where option

`-f`

Do *not* fail on error

`-r`

Messages not in the same order

`-v`

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=lsm 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
f2.grib1
    comparing totalLength as long
    comparing editionNumber as long
    comparing section1Length 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 stepRange f1.grib1 f2.grib1
-- GRIB #1 -- shortName=lsn paramId=172 stepRange=3
  levelType=sfc level=0 packingType=grid_simple
  gridType=reduced_gg --
string [stepRange]: [3] != [6]
```

- 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 --
string [stepRange]: [3] != [6]
long [startStep]: [3] != [6]
long [endStep]: [3] != [6]
long [validityTime]: [1500] != [1800]
```


`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 [P1]: [3] != [6]
```

- 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 [section1Length]: [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` – order-independent compare

- There are many errors if two files are compared which contain the same messages but in a different order

```
> grib_compare -f -H f1.grib1 f2.grib1
...
## ERRORS SUMMARY #####
##
## Summary of different key values
## indicatorOfParameter ( 6 different )
## level ( 7 different )
##
## 10 different messages out of 12
```

*By default
grib_compare
assumes messages
are in the same
order*

- To compare messages when they are not in the same order, use the `-r` option – this is VERY time expensive

```
> grib_compare -r -f -H f1.grib1 f2.grib1
```

`grib_compare` – comparing data values

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

`-A absolute_error`

Use absolute error as tolerance

`-R key=rel_error,...`

Use relative error as tolerance for **key**

`-P`

Use packing error as tolerance

`-T factor`

Compare data values using tolerance specified in options `-A`, `-R`, `-P` multiplied by an integer **factor**

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

Practical: using grib_compare

1. Use `grib_compare` to compare the headers of the GRIB messages contained in the files `file1.grib` and `file2.grib`
 - Use the “-H” option to restrict the comparison to the headers only
 - 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 “-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 `file3.grib` and `file4.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 value
<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**

`grib_to_netcdf` – convert to NetCDF

- Use `grib_to_netcdf` to convert GRIB messages to NetCDF
- Input GRIB fields must be on a regular grid
 - `typeOfGrid=regular_ll` or `regular_gg`
- Options allow user to specify
 - the NetCDF data type:
 - `NC_BYTE`, `NC_SHORT`, `NC_INT`, `NC_FLOAT` or `NC_DOUBLE`
 - `NC_SHORT` is the default
 - either classic (NetCDF 3) or NetCDF 4 file format
 - the reference date
 - default is 19000101
- Used in the MARS web interface and the public Data Servers to provide data in NetCDF files

grib_to_netcdf – usage

```
grib_to_netcdf [options] grib_file grib_file ...
```

Options

- `-o output_file` Output netCDF file
- `-R YYYYMMDD` Use `YYYYMMDD` as reference date
- `-D NC_DATATYPE` NetCDF data type
- `-k kind` Kind of file to be created:
 - 1 → netCDF classic file format
 - 2 → netCDF 64 bit classic file format (Default)
 - 3 → netCDF-4 file format
 - 4 → netCDF-4 classic model file format
- `-T` Do not use time of validity.
- `-u dimension` Set `dimension` to be an unlimited dimension
- `-f` Do *not* fail on error

...

grib_to_netcdf – examples

- To convert the fields in file.grib1 to NetCDF

```
> grib_to_netcdf -o out.nc file.grib1
```

```
grib_to_netcdf: Version 2.2.0
```

```
grib_to_netcdf: Processing input file 'file1.grib1'.
```

```
grib_to_netcdf: Found 1 GRIB fields in 1 file.
```

```
grib_to_netcdf: Ignoring key(s): method, type, stream, refdate, hdate
```

```
grib_to_netcdf: Creating netCDF file 'out1.nc'
```

```
grib_to_netcdf: NetCDF library version: 4.3.2 of May 10 2016 11:12:41 $
```

```
grib_to_netcdf: Creating large (64 bit) file format.
```

```
grib_to_netcdf: Defining variable 't2m'.
```

```
grib_to_netcdf: Done.
```

```
> ls -s out.nc
```

```
160 out.nc
```

grib_to_netcdf – examples

- To convert the fields in file.grib1 to NetCDF with data type set to **NC_FLOAT**

```
> grib_to_netcdf -D NC_FLOAT -o out.nc file.grib1
grib_to_netcdf: Version 2.2.0
grib_to_netcdf: Processing input file 'file1.grib1'.
grib_to_netcdf: Found 1 GRIB fields in 1 file.
grib_to_netcdf: Ignoring key(s): method, type, stream, refdate, hdate
grib_to_netcdf: Creating netCDF file 'out1.nc'
grib_to_netcdf: NetCDF library version: 4.3.2 of May 10 2016 11:12:41 $
grib_to_netcdf: Creating large (64 bit) file format.
grib_to_netcdf: Defining variable 't2m'.
grib_to_netcdf: Done.
```

```
> ls -s out.nc
316 out.nc
```

Output NetCDF file is about twice the size



Practical: modifying GRIB messages

1. The file file1.grib1 contains parameters T and Z on five pressure levels.
 - Use `grib_copy` to create two files, one containing all the pressure levels for parameter T, the other for Z. Check the content of the new files with `grib_ls`
 - Repeat but output the messages so the levels in the new files are in increasing numerical order
2. Use `grib_set` to change the date and time to 00UTC on 28 February 2017 for all messages in file1.grib1
 - Repeat but change the date and time for T at 500hPa **only**
 - Repeat so that T at 500hPa **only** is written to the output file
3. Use `grib_to_netcdf` to convert the GRIB messages in file2.grib1 to NetCDF.
 - Try with both the default data type (`NC_SHORT`) and `NC_FLOAT`. Check the data values in each case with `ncdump`.
 - Repeat but set the Reference date to 22 February 2017 and compare the time variable with previous results
4. Use `grib_to_netcdf` to convert the GRIB messages in file3.grib1 to NetCDF.
 - What happens ... and why ?

Extra practicals

1. Use `grib_copy` to split `file1.grib1` into separate files for each parameter/level combination
 - Create files named `t_500.grib1`, `z_500.grib1`, etc
2. An SST field has been created by masking the Soil Temperature at Level 1 (STL1) with the Land-Sea Mask and is included with other messages in the file `surface.grib1`
 - Use `grib_set` to change the parameter for the field from STL1 to SST and level type to 'surface'
 - Be careful not to change the other parameters !
 - Repeat with each different message output to a separate file
3. Use `grib_to_netcdf` to convert the data in `file4.grib1` to NetCDF
 - What happens ?
 - Follow the hint and try again !
 - Inspect the content with `ncdump`