loop_wrapper Documentation

Release 3.5.0

Thomas Lavergne

Apr 17, 2020

Contents:

CHAPTER 1

The Basics

`loop_wrapper` is a job-control tool to make it easier to loop through datetime ranges. `loop_wrapper` can be instructed to loop through a range of dates and spawn a command you specify for each date in the range. The command can be an excutable or a shell command.

Here is a first basic example:

```
[$] loop_wrapper --quiet 19900101 19900105 echo "here is a date {d:%Y/%m/%d}"
here is a date 1990/01/01
here is a date 1990/01/02
here is a date 1990/01/03
here is a date 1990/01/04
here is a date 1990/01/05
```

The `loop_wrapper` command-line has always such structure.

```
[$] loop_wrapper [optional flags] <START-DATE> <STOP-DATE> <COMMAND>
```

In the first basic example above, `--quiet` is a flag that reduces verbosity. `<START-DATE>` is 01/01/1990, `STOP-DATE` is 5 days later 05/01/1990, and the `<COMMAND>` is a Bash `echo` invocation. The most important part of this command is the `{d:%Y/%m/%d}` construct. This `{d:}` directive tells `loop_wrapper` how to print the datetime at each date in the range. There must be at least one such `{d:}` construct in `<COMMAND>`.

For the second example, consider you have an executable `process` (script or compiled it does not matter). Among other parameters, `process` can take `-d DDMMYYYY` and will then do some processing for that day. Let's further assume that `process` also takes a `-i` (input)

```
[$] loop_wrapper 20150227 20150302 process -i /path/to/inputdir -d {d:%Y%m%d}
CMD is process -i /path/to/inputdir -d {d:%Y%m%d}
Serial run process -i /path/to/inputdir -d {d:%Y%m%d} from 2015-02-27 00:00:00 to␣
↪2015-03-02 00:00:00
do  (process -i /path/to/inputdir -d 20150227)
do  (process -i /path/to/inputdir -d 20150228)
do  (process -i /path/to/inputdir -d 20150301)
do  (process -i /path/to/inputdir -d 20150302)
Done
```

Since we did not specify `--quiet`, we get to see more information from `loop_wrapper`, including what `COMMAND` (`CMD`) is (1st line), that a *serial* run is prepared from 27/02/2015 to 02/03/2015 (2nd line), the commands that the shell is instructed to run in turn (the *do (. . .)* lines). And finally that we are *Done*.

CHAPTER 2

Running on several CPUs

It is easy to switch from a *serial* run to a *parallel* one, just use the `--cpu all` option. `loop_wrapper` will then divide the range of dates to be processed into smaller chunks, and distribute the chunks to the available CPUs. A Python multiprocessing *Pool* is used to balance the load.

You can also control the number of CPUs to be used with `--cpu N`. If *N* is a positive number, it indicates the number of CPUs to use. If *N* is a negative number, it indicates the *number of CPUs to save*. `--cpu -2` will use all available but *2* CPUs.

Warning: The use of parallel runs with the `--cpu` option **does not guarantee** the order in which the dates are processed.

CHAPTER 3

The {d:} construct

A `{d:format}` construct is needed for the `<COMMAND>` to work in `loop_wrapper`. All the Python datetime strftime constructs are allowed. For example `{d:%Y%m%d}` (*20150216*), `{d:%Y%j}` (*2015047*), `{d:%Y/%m/}` (*2015/02/*). This gives full freedom to format the date as required by the processing command.

Note: `{d:}` (no explicit format given) is equivalent to `{d:%Y%m%d}`.

CHAPTER 4

Date looping controls

4.1 Datetime stepping

`loop_wrapper` supports stepping different lengths and different units. Datetime stepping is controlled by the `--every` flag, that defaults to `--every 1d` for steps of 1 day. The general format is `--every Nu` where *N* is a positive integer, and *u* a unit. The supported units are:

Table 1: Units for `--every`

u	Time Unit
d	day
m	month
Y	year
H	hour
M	minute
S	second
w	weekly

We provide two examples with `--every`

```
[$] loop_wrapper --every 1m 20150201 20150501 process -i /path/to/inputdir/{d:%Y/%m/}
CMD is process -i /path/to/inputdir/{d:%Y/%m/}
Serial run process -i /path/to/inputdir/{d:%Y/%m/} from 2015-02-01 00:00:00 to 2015-
↪05-01 00:00:00
do  (process -i /path/to/inputdir/2015/02/)
do  (process -i /path/to/inputdir/2015/03/)
do  (process -i /path/to/inputdir/2015/04/)
do  (process -i /path/to/inputdir/2015/05/)
Done

[$] loop_wrapper --every 6H 2015041006 2015041118 process -H {d:%H}
CMD is process -H {d:%H}
Serial run process -H {d:%H} from 2015-04-10 06:00:00 to 2015-04-11 18:00:00
```

(continues on next page)

(continued from previous page)

```
do  (process -H 06)
do  (process -H 12)
do  (process -H 18)
do  (process -H 00)
do  (process -H 06)
do  (process -H 12)
do  (process -H 18)
Done
```

4.2 Backwards looping

The default for `loop_wrapper` is to loop from `<START-DATE>` to `STOP-DATE` forward in time. You can also instruct *backward* looping with the `--backwards` flag.

Note: `--backwards` cannot be used with *parallel* runs (`--cpu`) since the execution order is then not guaranteed.

Note: Even with `--backwards`, the oldest date should be in `<START-DATE>` and the newer date in `STOP-DATE`.

4.3 <START-DATE> and <STOP-DATE>

The `<START-DATE>` and `<STOP-DATE>` arguments can be specified with several formats as described in the table below.

Table 2: <START-DATE> and <STOP-DATE> format

Format	Default
YYYYMMDD	At 00:00:00
YYYYMMDDHH	At HH:00:00
YYYYMMDDHHMM	At HH:MM:00
YYYYMMDDHHMMSS	At HH:MM:SS
YYYYMM	On 01/MM/YYYY
YYYY	On 01/01/YYYY
TODAY	At 00:00:00
YESTERDAY	At 00:00:00

CHAPTER 5

Wildcard expansion

It is sometimes not enough to pass a datestring as a parameter to a `process` script. `loop_wrapper` also gives the possibility to use a wildcard expansion to find files or directories with provided patterns.

5.1 Using wildcards to find files

For example, consider a directory tree with content

```
./
├── 2005
│   ├── tst_A_20051228.tar
│   ├── tst_A_20051229.tar
│   ├── tst_A_20051230.tar
│   ├── tst_A_20051230.zip
│   ├── tst_A_20051231.tar
│   ├── tst_B_20051228.tar
│   ├── tst_B_20051229.tar
│   ├── tst_B_20051230.tar
│   ├── tst_B_20051231.tar
│   ├── tst_C_20051228.tar
│   ├── tst_C_20051229.tar
│   ├── tst_C_20051230.tar
│   └── tst_C_20051231.tar
├── 2006
│   ├── tst_A_20060101.tar
│   ├── tst_A_20060102.tar
│   ├── tst_AA_20060101.tar
│   ├── tst_AA_20060102.tar
│   ├── tst_B_20060101.tar
│   ├── tst_B_20060102.tar
│   ├── tst_C_20060101.tar
│   └── tst_C_20060102.tar
```

The command below will un-tar all the `tst_?_.tar` files with date ranging from 30/12/2005 to 02/01/2006:

```
[$] loop_wrapper 20051230 20060102 'tar -xf [./{d:%Y}/tst_?_{d:}.tar]'
CMD is tar -xf [./{d:%Y}/tst_?_{d:}.tar]
Serial run tar -xf [./{d:%Y}/tst_?_{d:%Y%m%d}.tar] from 2005-12-30 00:00:00 to 2006-
↪01-02 00:00:00
do (tar -xf ./2005/tst_A_20051230.tar)
do (tar -xf ./2005/tst_B_20051230.tar)
do (tar -xf ./2005/tst_C_20051230.tar)
do (tar -xf ./2005/tst_A_20051231.tar)
do (tar -xf ./2005/tst_B_20051231.tar)
do (tar -xf ./2005/tst_C_20051231.tar)
do (tar -xf ./2006/tst_A_20060101.tar)
do (tar -xf ./2006/tst_B_20060101.tar)
do (tar -xf ./2006/tst_C_20060101.tar)
do (tar -xf ./2006/tst_A_20060102.tar)
do (tar -xf ./2006/tst_B_20060102.tar)
do (tar -xf ./2006/tst_C_20060102.tar)
Done
```

The command-line above introduces new constructs that are necessary for using wildcard expansion in `loop_wrapper`. First, the `<COMMAND>` is enclosed in single-quotes `'`. This prevents the shell to perform its own wildcard expansion before the `loop_wrapper` event starts.

Second, the part of `<COMMAND>` that contains wildcards (in this case `?`) and requires expansion is enclosed in square brackets `[]`. Wildcard expansion will be attempted only for parts of the `<COMMAND>` enclosed in `[]`. There can be several of these in the same `<COMMAND>`.

When desiging `loop_wrapper` calls with wildcards, it is usefull to consider the order in which the operations are processed. First a command is prepared with substitution of all the `{d:format}` constructs, then for each such command, wildcard expansion is performed to find files matching the pattern. New commands are created and started for each new file.

The syntax for wildcard expansion is that of the Python glob module, allowing many wildcards such as `*` (any characters, any number), `?` (any character, once), `[0-9]` (any character in the range, once), etc...

5.2 Re-using file names with {f:}

Each time a wildcard expansion is successfull (finds an existing file), the full path of the file is stored and can be re-used on the same command with constructs `{f:}` (full path) and `{F:}` (basename).

For example, consider you have a tool `convert FILE1 FILE2` and many files to *convert* arrange in a hierarchy of sub-directories below `in/`, the following command will process some of them and store the result in another set of sub-directories below `out/`:

```
loop_wrapper 20150216 20150315 'convert [in/{d:%Y/%j}/ex_*_{d:%Y%m%d}.nc] out/{d:%Y/
↪%m/}{F:}'
```

CHAPTER 6

Verbosity and reporting

Several options allow to control the level of verbosity and reporting from `loop_wrapper` runs.

`--quiet` removes all output by `loop_wrapper` and only the output from the processing `<COMMAND>` are printed.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search