



# Running tasks in parallel: ParallelTaskCollection

Riccardo Murri <[riccardo.murri@uzh.ch](mailto:riccardo.murri@uzh.ch)>

*S3IT: Services and Support for Science IT*

University of Zurich

## Basic use of ParallelTaskCollection

```
from gc3libs.workflow \
    import ParallelTaskCollection

class MyTasks(ParallelTaskCollection):
    # ...
    def __init__(self, ...):
        app1 = AnApp(...)
        app2 = AnotherApp(...)
        # ...
        appN = YetAnotherApp(...)
        ParallelTaskCollection.__init__(
            self, [app1, app2, ..., appN])
```

A `ParallelTaskCollection` runs a list of tasks, submitting at the same time as many as possible.

## Basic use of ParallelTaskCollection

```
from gc3libs.workflow \
    import ParallelTaskCollection

class MyTasks(ParallelTaskCollection):
    # ...
    def __init__(self, ...):
        app1 = AnApp(...)
        app2 = AnotherApp(...)
        # ...
        appN = YetAnotherApp(...)
        ParallelTaskCollection.__init__(
            self, [app1, app2, ..., appN])
```

Initialize a `ParallelTaskCollection`  
with a list of tasks to run.

## Running tasks in parallel

```
class MyScript (SessionBasedScript):  
    # ...  
    def new_tasks(self, extra):  
        tasks_to_run = [  
            MyTasks(...)  
        ]  
        return tasks_to_run
```

You can then run the entire collection by returning it from `new_tasks()`, or by using it as a step in an outer `SequentialTaskCollection`.

## Detour: random colors

**ImageMagick** allows specifying colors also by the syntax `xc:#rgb` where *r*, *g*, and *b* are 2-digit hexadecimal numbers ranging from 00 to ff.

For example, `#ffa500` is the “orange” color.

You can generate these color specifications randomly from Python by combining **string formatting** and the **randint** function:

```
def random_color():  
    from random import randint  
    r = randint(0, 255)  
    g = randint(0, 255)  
    b = randint(0, 255)  
    color = ("xc: #%02x%02x%02x" % (r, g, b))  
    return color
```

**Exercise 9.A:** Write a `RandomlyColorize` task collection class (subclass of `ParallelTaskCollection`) that is initialized with two parameters: an image file name `img` and a number `N`.

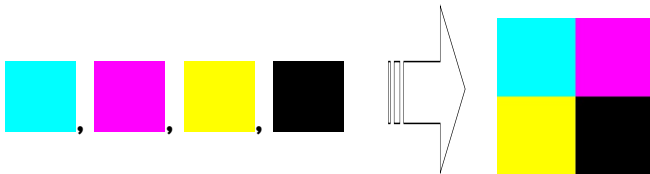
The `RandomlyColorize` collection then consists of `N` instances of the `ColorizeApp`, each initialized with the same image file name `img` and three random colors.

**Exercise 9.B:** Modify the `RandomlyColorize` class from Exercise 9.A to:

1. take an additional parameter `tolerance` (a number from 0.0 to 100.0)
2. upon termination of all tasks in the collection, set the collection's exit code to 0 (success) if at the percentage of failed jobs is less than `tolerance`; set the exit code to 1 otherwise.

## Detour: image montage

```
$ montage \  
  cyan.jpg magenta.jpg yellow.jpg black.jpg \  
  -tile 2x2 -geometry +0+0 cmyk.jpg
```





## Write your own “Warholize” workflow!

**Exercise 9.C:** Write a `WarholizeScript` session-based script.

The script takes any number of image file names as arguments; each image undergoes the following processing steps:

1. Conversion to grayscale;
2. From the grayscaled version,  $N^2$  randomly-colored new images are formed;
3. These  $N^2$  tiles are arranged in a  $N \times N$  grid, which is the final result of the processing pipeline.

A command-line option `--size` allows one to set the value of  $N$ , with default  $N = 2$ .