# MY BACKUP MANUAL

António Manuel Dias

2021/02/17

## FOREWORD

The purpose of this program is to ease the backup and restore of a group of directories to/from a different location, using an external utility like the unix programs `rsync` or `cp`. This is accomplished in the following manner:

1. Read a *backup profile* from a user defined configuration file.
2. Iterate over the group of directories in the profile issuing the command to copy them from the base path to the destination directory.

It is possible to have different profiles in the user configuration file, with different backup strategies and targets. For an overview of several possible use cases, check the section **USE CASES** at the end of this manual.

The program depends on Python 3 and the availabity of the external utility you intend to use. It was thought to be used at the command line (terminal) of a Linux system, but it will probably work seamlessly in any unix like system like xBSD or MacOS, provided Python 3 is installed. It may also be possible to make it work under MS Windows, if the user knows enough system administration to tweak it to do that. Although I have not tested it in any of those systems, some tips to install and use the program under Windows are given in the subsection **Usage in MS Windows** of the **USAGE** section.

Changes history:

- 2.2: Added option to edit the configuration file
- 2.1: Added option to list backup profiles Added option to display configuration file Composite profiles can now only be composed of simple profiles, thus preventing 'infinite loop' profiles
- 2.0: Complete rewrite in Python. Added possibility of multiple configuration profiles in a separate file.
- 1.0: Original bash script.

## INSTALLATION

The instructions below are for installation on a modern Linux system. They may work on other modern unix-like systems like BSD derivatives, including MacOS, but that has not been tested and may require some tweaking. Try it at your own risk. For MS Windows installation, refer to the subsection **Usage in MS Windows** in the **USAGE** section.

1. Download and unzip the program's compressed file in a directory of your choosing. To download the most recent version, check AMMDIAS site.
2. Open a terminal in the directory where the program was uncompressed and run one of the following scripts to install locally or globally:
   - Local installation:

     ```
     $ bash local_install.sh
     ```

     This will install the program for the current user only and is suited for single-user systems. The program will be installed on a directory under `$HOME/.local/lib` and a symbolic link `$HOME/.local/bin/mybkp` will be created to allow simple execution. The sample configuration file will be copied to `$HOME/.config/mybkp_profiles`.

     On most Linux systems the `$HOME/.local/bin` directory will be inserted in the execution PATH, if it exists. If that is not your case, you must make sure to insert it.

   - Global installation:

     ```
     $ sudo bash global_install.sh
     ```

     This will install the program for all the system's users and should be the option if it is installed in a multi-user system. The program will be installed under `/usr/local/lib` and a symlink `/usr/local/bin/mybkp` will be created. The configuration file must be copied by each user to a suitable location (see section below).

   - Manual installation:

     Copy the uncompressed program directory to a location of your choosing and create a symbolic link to the program (`mybkp.py`) anywhere in your PATH.
3. Test that the installation was successful with the command:

```
$ mybkp --help
```

Before using the program, you must configure at least one backup profile. Read the following section for instructions on how to accomplish that.

## CONFIGURATION

This section will focus on the syntax of the configuration file for the program. For actual usage examples, please refer to the last section, **USE CASES**. The calling syntax and program options may be found in the next section, **USAGE**.

The program will look for the backup profile to execute in a file in one of these locations:

- The path passed to the program with the `-c`/`--config` options
- `$HOME/.mybkp_profiles`
- `$HOME/.config/mybkp_profiles`

If you installed the program globally or manually, do not forget to create that file with a text editor and save it to one of those locations. To copy the sample configuration file from the global installation to your user profile, use these commands:

```
$ mkdir -p ~/.config
$ cp /usr/local/lib/mybkp-2.2.1/mybkp_profiles ~/.config/
```

If you wish to pass the configuration file location to the program manually at calling time, you may place it anywhere you want.

### Configuration file syntax

The **configuration file** is a simple text file, divided in *sections* started by a *name* between square brackets and followed by a number of *name*/*value* pairs. Empty lines or lines started with a semi-colon are ignored.

A **backup profile** is a *section* that has a name chosen by the user and that must contain four *name*/*value* pairs:

- `command`: the command to perform the backup, including all the command's options; this must be a system recognizable command that is called with two positional arguments in the form `command <source> <destination>` (like `cp`, `scp` or `rsync`);
- `base`: the parent directory where the directories to backup reside (e.g. your home directory);

- `backup`: the directory where the backup will reside (e.g. an external disk drive or network storage device);
- `directories`: a comma separated list of directories, below the *base* directory, to backup.

A **composite backup profile** is composed of a list of previously defined profiles, which will be executed in series. It has a single name/value pair:

- `profiles`: a comma separated list of previously defined profile names.

Below is an example of a configuration profile with two normal and one composite profile:

```
; Example configuration file

[documents]
command: cp -auv --strip-trailing-slashes
base: /home/antonio
backup: /media/antonio/BKP_DISK
directories: Documents, Templates

[media]
command: cp -auv --strip-trailing-slashes
base: /home/antonio
backup: /media/antonio/BKP_DISK
directories: Music, Pictures, Videos

[all]
profiles: documents, media
```

Note that the `command` value must include all the options and the `base` and `backup` values must be full paths. The `directories` value is a list of directories directly under the base and backup directories. If you wish to backup a subdirectory only, you must include it explicitly. For example, `Pictures/Photos` will backup that directory but not others under `Pictures` like `Pictures/Wallpapers` unless explicitly included also.

A value may refer to the value of another section with the syntax `${SECTION:VALUE}`. For example, the `media` section above could have been written this way:

```
[media]
command: ${documents:command}
base: /home/antonio
backup: /media/antonio/BKP_DISK
```

```
directories: Music, Pictures, Videos
```

This may simplify the writing of complex configuration files and avoid typing errors. An initial section of *variables* is possible using this feature. The example configuration file above could have been written this way:

```
[VAR]
cmd: cp -auv --strip-trailing-slashes
home: /home/antonio
bkp: /media/antonio/BKP_DISK

[documents]
command: ${VAR:cmd}
base: ${VAR:home}
backup: ${VAR:bkp}
directories: Documents, Templates

[media]
command: ${VAR:cmd}
base: ${VAR:home}
backup: ${VAR:bkp}
directories: Music, Pictures, Videos

[all]
profiles: documents, media
```

## USAGE

This program is to be used from the command line, with the following syntax:

```
$ mybkp [option [option ...]] [profile [profile ...]]
```

This supposes that `mybkp` is a symbolic link in your execution PATH to the actual `mybkp.py` program location and that the configuration file is in one of its default locations, `~/.mybkp_profiles` or `~/.config/mybkp_profiles`.

If that is not the case, replace `mybkp` with the actual path to the program and pass the location of the configuration file with the `-c` option (see **Options** section below). For example, if the program is located in the diretory `programs/mybkp-2.2.1` under the user's home directory, he would call it like this:

```
$ ~/programs/mybkp-2.2.1/mybkp.py [options [option ...]] \
                                  [profile [profile ...]]
```

or, if the program was not executable:

```
$ python3 ~/programs/mybkp-2.2.1/mybkp.py \
          [options [option ...]] [profile [profile ...]]
```

## Profiles

The only positional arguments accepted by the program are the names of the backup profiles to be executed. You may pass any number of profiles but their names must match exactly the names on the configuration file. If no profile name is passed, the program will try to execute a profile named `default`, if it exists.

Examples:

```
$ mybkp documents
```

Will try to read and execute the profile named `documents` in the configuration file.

```
$ mybkp documents media
```

Will do the same for the profiles `documents` and `media`.

```
$ mybkp
```

Will try to read and execute the profile named `default` in the configuration file.

## Options

This is a list of all the optional arguments accepted by the program:

- `-h` / `--help`, `--copyright`, `--warranty`, `--version`

  Display the program's online help page, copyright, warranty or version information and exit the program. Example:

  ```
  $ mybkp --help
  ```

- `-b` / `--backup_directory` *name*

  Add `name` to the end of the profile's backup directory path. Must be followed by the name to add. Example:

  ```
  $ mybkp -b 2020-August documents
  ```

  Will copy the `documents` backup profile directories to the `2020-August` directory under the profile's backup directory.

- `-c` / `--config` *configuration file*

  Use the configuration file provided instead of the default one. Must be followed by the (absolute or relative) path to the file. Examples:

  ```
  $ mybkp -c test/alt_config_file test_profile
  ```

  Will execute the `test_profile` profile in the `alt_config_file` configuration file that is located in the `test` directory.

- `-e` / `--edit_config`

  Try to edit the configuration file using the system's default editor. The program will scan the 'VISUAL' and 'EDITOR' environment variables, by this order, to find the the default editor command. If none of these variables is set the program will display an error message and exit.

  ```
  $ mybkp -e
  ```

- `-i` / `--include_destination_directories`

  By default, the copy command's destination argument is the backup directory. This option will add the profile's directories to the destination argument of the command, as required by some programs (like `rsync`). Example:

  ```
  $ mybkp -i documents
  ```

  If the *command* was defined in the `documents` profile as `rsync -aPhv`, the *directories* were defined as `Documents` and `Templates`, the *base* was defined as `/home/antonio` and the *backup* as `/media/antonio/BKP_DISK`, the commands issued by the program would be:

  ```
  rsync -aPhv /home/antonio/Documents/ /media/antonio/BKP_DISK/Documents
  rsync -aPhv /home/antonio/Templates/ /media/antonio/BKP_DISK/Templates
  ```

  Without this option, the commands would be:

  ```
  rsync -aPhv /home/antonio/Documents/ /media/antonio/BKP_DISK
  rsync -aPhv /home/antonio/Templates/ /media/antonio/BKP_DISK
  ```

- `-l` / `--list_profiles`

  Display a list of valid backup profiles on the configuration file and exit the program. Example:

  ```
  $ mybkp -l
  ```

- `-n` / `--no_act`

Displays the commands that would be issued by the program instead of actually executing them. Example:

```
$ mybkp -n documents
```

- `-p` / `--print_config`

  Display a verbatim copy of the configuration file and exit the program.

  ```
  $ mybkp -p
  ```

- `-r` / `--restore`

  Swaps the *base* and *backup* directories in the copy command's arguments, copying the profile's defined *directories* from the *backup* to the *base* directory, efectively restoring a backup to the original location. Example:

  ```
  $ mybkp -r documents
  ```

## Usage in MS Windows

As mentioned previuosly, *My Backup* was meant to be used in Linux systems but, as Python is ported to every major operating system, it may be possible, with some tweaking, to run it on other systems. In this subsection I will be giving some tips to what could be done to make it work in MS Windows. Notice that **THIS HAS NOT BEEN TESTED** so, proceed at your own risk :)

1. Start by downloading and installing the most recent version of the Python language to your system.

   When prompted, answer yes to the question about adding Python to your system PATH.

2. Download the most recent version of *My Backup* from AMMDIAS site and uncompress it in a directory of your choosing.

3. Inside that directory, create a configuration file with a text editor. If you create it from scratch, you may use the Windows Notepad. If you wish to start with the sample configuration file you will need another simple text editor like Notepad++ or even the editor from the Python IDE that you just installed (IDLE), because the original file uses Unix line endings that Notepad does not understand (the most recent versions of Notepad don't have this problem).

   Some notes about the configuration file:

- The *base* and *backup* directories must have their full path, including the drive;
- The *command* must be a Windows program, like `xcopy`.

4. Create a *batch file* for each backup and restore operation. This way you may launch each operation just clicking on those files, without the need to use the command line.

Sample configuration file:

```
; Sample My Backup configuration file for MS Windows
; Last updated: 2021-02-17 (AMD)

[documents]
command: xcopy /d /e /i /f /h /r /k /b /j
base: C:\Users\antonio
backup: E:\
directories: Documents, Desktop

[media]
command: xcopy /d /e /i /f /h /r /k /b /j
base: C:\Users\antonio
backup: E:\
directories: Music, Pictures, Videos

[all]
profiles: documents, media
```

Explanation:

There are three profiles, one to backup the `Documents` and `Desktop` folders of the user `antonio`, another to backup the `Music`, `Pictures` and `Videos` folders, and the last one is a composite profile that backups everything.

Note that both the *base* and *backup* directories have their full paths: the first is Antonio's personal folder and the other is the root of drive `E:` (which may be a separate drive, an external USB drive or even a network mapped drive).

Both profiles use the `xcopy` command with the same flags. For their meaning, check the MS Windows documentation.

Sample batch file to execute the `documents` profile:

```
:: Sample Windows batch file to execute My Backup
:: on 'documents' profile
```

```
"C:\Program Files\Python\python.exe" ^
  C:\Users\antonio\Programs\mybkp-2.2.1\mybkp.py ^
  --config mybkp_profiles.txt documents
```

Save this file in the folder where you uncompressed *My Backup* with a name with the extension ".bat". If you use Notepad to create the file you must surround the name in double-quotes, otherwise the file will be saved with a ".txt" extension. For example, this batch file could be named `"backup_documents.bat"`.

On this batch file I assume that the Python executable is in

`C:\Program Files\Python\python.exe`

(you must check the actual location), that *My Backup* is installed at

`C:\Users\antonio\Programs\mybkp-2.2.1`

(you must change it to where you uncompressed it) and that the backup configuration file is called `mybkp_profiles.txt` (change it to the name you used).

You must now create a similar batch file for each of the profiles. The restore batch files are identical, with just the addition of the `--restore` argument.

To run the program you just need to open the *My Backup* program folder and click on the desired batch file. You may even create links on the desktop or on the Start Menu, if you prefer.

## USE CASES

The way this program works matches closely the way I maintain my backups. It was made for that particular purpose: simplify and organize the backup of my personal data, trying to avoid data loss due to mistakes or hardware failure.

Several backup strategies were invented over the years and I tried a bunch of them, but the one that makes me most comfortable is to have the backup replicate exactly the data structure I have at my computers. That is the reason I use tools like `cp` and `rsync` and not backup or version control programs. This way, if I make a mistake and change a single file or delete a directory by acccident I can just resort to one of the backups and copy that file or directory back to my system, without worrying about reverting other changes I could have made or nit-picking through old versions to find what I want. Of course, this is what fits **me** – other people will have different needs and preferences.

The very first thing one should do when planning a backup strategy is to identify what needs to be backed up. In my case, in order of importance, this is what I

found:

- In my personnal computer:
  1. Personnal documents and projects: legal documents, finance information, source code and private stuff like emails and messages. Also, photos and personnal videos.
  2. Configuration preferences: *dot-files* (`bashrc`, `vimrc`, etc.), and system setup information or instructions.
  3. Multimedia: books, music, movies...
- In my server:
  1. Server data, like web server files and databases.
  2. Server configuration, systemd scripts, SSL keys, etc.

I do not backup whole systems or programs: I know I can restore them from scratch in a few hours at most. For example, if my personnal computer burns I am not worried about the programs or operating system, because that is easy to get from the Internet and install in a new computer. I am **very** concerned about losing any personal data I have on that computer, because if I don't have an updated backup there is no other place I can get it.

The second step in planning a backup is to know where it will reside. If you are paranoid like me about loosing your data, you will want to have more than one separate location. I just do not use *cloud* backup, for privacy concerns – I barely trust my own systems, even encrypted as they always are, there is no way I would trust other people's computers (and that is what *the cloud* really is).

The use cases described below are mostly derived from these needs. I will use my user name (`antonio` or `adias`) on all examples, just because I had to choose one.

## Preliminary planning

Before writing the profile configuration file, there is some information that needs to be gathered. Let's do it now.

Start by identifying the directories you want to backup and the topmost directory of them all, that will serve as *base*. In a modern Linux system, all directories will probably be under your home directory. In this case we will use these:

- `/home/antonio/Documents`, the personnal documents folder
- `/home/antonio/Pictures/Photos`, my personnal photos
- `/home/antonio/Projects`, source code of my projects
- `/home/antonio/Work`, professional work I keep at home

The topmost directory is the home directory, `/home/antonio` and that will be the *base*. The *directories* will be `Documents`, `Pictures/Photos`, `Projects` and `Work`. If you don't want to completely configure a system when you have to install it from scratch, you should probably also backup your configuration files. Let's say these are what I want to keep:

- `/home/antonio/.profile`
- `/home/antonio/.bashrc`
- `/home/antonio/.bash_aliases`

The problem is that these files all reside in a directory (the home directory) where there is also a lot of files and directories I do not want to bakup. The solution I found is to put all those files in a single directory and then make symbolic links to them in their previous location. So, all those files reside in `/home/antonio/.config/dotfiles` and I that is the directory I have to add to the configuration file.

Now, let's say the backup is to reside in an external USB drive partition named `BKP_DISK`. In Ubuntu Linux, external USB drives are automatically mounted when first accessed in a directory named after the partition name under the `/media/username` directory. So, in our example the *backup* location will be `/media/antonio/BKP_DISK`.

Finally, we need to decide on what tool we will use to copy the data to the backup location. Let's sart with the basic, `cp`, and decide what options we need to include:

- `-a` / `--archive`: this option will ensure that symbolic links are copied (instead of the target of those links), that all file attributes will be preserved if possible and that the copy will be recursive, i.e. will include subdirectories;
- `-u` / `--update`: this option will make subsequent copies faster, as it will only copy files that are missing from the backup or that have been changed after the last backup;
- `-v` / `--verbose`: to print on screen the changes that are being made;
- `--strip-trailing-slashes`: to remove trailing slashes from source directories, meaning that they need to be copied also, not just their contents.

With this information, we can now proceed to the use cases.

## Single profile

This is the case where you have only one backup location, like and external USB drive or a folder on a `ssh` accessible server, and a single group of directories to backup. Here you should probably configure a single profile named `default`,

which will save you from typing the profile name at the command line every single time you want to update the backup.

The information colected above is all we need to make the configuration file:

```
; Example 1

[default]
command: cp -auv --strip-trailing-slashes
base: /home/antonio
backup: /media/antonio/BKP_DISK
directories: Documents, Pictures/Photos, Projects, Work,
             .config/dotfiles
```

To make or update the backup, all we need to do now is conect the external drive to the computer, open it and, in a terminal, call `mybkp`. First, we should always test the program in *dry-run* mode (see **USAGE** above), to test if the commands given are what we want:

```
$ mybkp -n
```

Then, if all seems ok, do the actual backup:

```
$ mybkp
```

To restore the backup completely, we would do:

```
$ mybkp -r
```

Or, if what we needed was to simply recover a single file or directory, just open the backup drive and copy them directly to our system.

## Remote backup location

If the backup is located on a remote machine, we cannot use `cp`. One solution is to use `scp`, to make the copies over a `ssh` connection. For this, we have to change the *program* and *base* values of the configuration file above:

```
; Example 2

[default]
command: scp -rpv
base: /home/antonio
backup: antonio@backup.home:/home/antonio/laptop-backup
directories: Documents, Pictures/Photos, Projects, Work,
             .config/dotfiles
```

`scp` will use your `ssh` configuration (if you have a remote backup server, you probably already know this). The arguments for the program are as follows:

- `-r`: recursive mode, to copy subdirectories;
- `-p`: to preserve file attributes;
- `-v`: to explain what is being done, like in `cp`.

There is one substantial difference between `cp` and `scp`: the latter will follow links and copy the actual file instead of the link. If that is not what you wish, a different program must be used, like `rsync` (see subsection below). If the `ssh` server is listening on a port other than the default, the port number must be included in the arguments. Example if the port is 2222:

```
command: scp -rpv -P 2222
```

The *backup* directory must include the remote user name and server name (or IP address) followed by a colon and then the directory path on the server. If you have the host configured in your local `ssh` configuration you may use the profile name instead. HOST profile names are a good thing, as they allow associating a host configuration (server name/address, login name, port) with a single name, thus simplifying the access to a remote server over `ssh` – check `ssh_config` man page for more information.

### Remote backup location using `rsync`

For my own setup, `cp` and `scp` have a big drawback, they cannot keep two directories really synchronized. This is because when updating a directory they only copy files and directories from the source to the destination, they will not delete the files or directories on the destination that you deleted or moved on the source. For example, if you have a backup of a directory and then rename a file on the source, when you update the backup you will then have two files: the old one and the copy of that file with the new name.

For some uses, this may be exactly what you want as you will keep old versions of files and directories, but in my case, where what I usually want is to keep exact copies of my systems' files in the backup, this doesn't work (I would have to manually delete old files, which would invalidate the purpose of the backup program).

`rsync` solves this problem and can be a drop-in replacement for `cp` or `scp` in the configuration of *My Backup*. It is not installed by default on Ubuntu Desktop (the Linux distribution I use) and probably the same happens on other Linux distributions, but is certainly available on all repositories. It is also natively available on xBSD and MacOS systems.

Taking the previous example, all you needed to use rsync instead of `scp` would be to replace the *command* value in the configuration profile:

```
; Example 3

[default]
command: rsync -aPhv --delete
base: /home/antonio
backup: antonio@backup.home:/home/antonio/laptop-backup
directories: Documents, Pictures/Photos, Projects, Work,
             .config/dotfiles
```

As `rsync` also uses `ssh`, the *backup* value remains the same. The options I used are as follows:

- `-a`: *archive mode*, which means recurse into subdirectories, preserve symbolic links, attributes, ownership, etc.;
- `-P`: display copy progress (useful for big copies);
- `-h`: display human readable file sizes (KB, MB, etc);
- `-v`: *verbose mode*, to explain what the program is doing
- `--delete`: delete files and directories from the destination directory that do not exist in the source directory.

There is one difference that must be taken into account when using `rsync`: this program expects the name of the directories in the destination argument. You can force this behaviour passing the `--include_destination_directories` or its short version `-i` to *My Backup*:

```
$ mybkp -i
```

This is also necessary when restoring the backup:

```
$ mybkp -ir
```

or

```
$ mybkp --include_destination_directories --restore
```

When using the `--delete` option with `rsync` extra care should be taken when doing a backup. If you accidentally delete a file or directory from the source and then do a backup, that file or directory will also be deleted from the backup. Before doing a backup, make sure that the original system is correct!

## Backups with multiple profiles

The main reason I rewrote *My Backup* was that I had different backups in different places. As the previous version kept the configuration on the script itself, I had to have a different script for each backup and that was becoming unmaintainable. Now, with the configuration on a separate file and the possibility of using completely different backup profiles with the same program, that problem is gone.

Let's look at some examples. The first supposes I have two backup locations: an external hard-drive, where I can keep a complete backup and a media server, where I just need the music, videos and photos. For this, I would write the configuration file below:

```
; Example 4

[VAR]
cmd: rsync -aPhv --delete
home: /antonio/home
media: Music, Pictures/Photos, Videos

[backup-disk]
command: ${VAR:cmd}
base: ${VAR:home}
backup: /media/antonio/BKP_DISK
directories: Documents, Templates, Projects, Work,
             ${VAR:media},
             .config/dotfiles

[media-server]
command: ${VAR:cmd}
base: ${VAR:home}
backup: media@media.home:/home/media
directories: ${VAR:media}
```

So, to make a full backup to the external drive I would call:

```
$ mybkp -i backup-disk
```

and to backup just the media files to the media server:

```
$ mybkp -i media-server
```

Things to notice on this configuration file:

- I used variables for the *command* and *base* values and also for the media directories: see the **CONFIGURATION** section to know how to use them.
- The directories value on the `backup-disk` profile are split over three lines. This is possible as long as the configuration parser can distinguish each extra line from the name of a value or the start of a section.

Notice also that I always pass the `-i` option to `mybkp`, because I am using `rsync` as the tool to copy files. If you don't want to keep writing that option or forget to use it, set an alias for the `mybkp` call in your `.bash_aliases` or `.bashrc` configuration file as follows:

```
alias mybkp='mybkp -i'
```

This way, the shell will replace `mybkp` with `mybkp -i` every time you call it.

If I feel the need to add another backup, I just have to add the corresponding section to the configuration file. For example, let's say I wish to backup the configuration files and scripts from my web server, that live on the directories `source` and `bin` of my user on that system, to my laptop. This is the section I would add to the previous configuration file:

```
; Example 4.1

[web-server]
command: ${VAR:cmd}
base: antonio@webserver.home:/home/antonio/
backup: ${VAR:home}/Projects/webserver/config
directories: source, bin
```

See that the *base* value is now the remote directory and the *backup* is a directory on the local system, and that I used a variable for the first part. Variables are just bits of text that will be replaced by their value when parsed by the program. The usage is the same as for the other profiles, even though the direction of the backup is different (from remote to local):

```
$ mybkp -i web-server
```

## Synchronizing multiple main systems

If you work on the same project or files in two different machines, say your home desktop and the one at work, you may wish to keep those files in sync. This may be accomplished with *My Backup*, if it is installed on both machines.

Let's say the transport from one machine to the other is done with a USB stick (with label `WORKDEV`) and the files you need are all on the `Work/Big Project`

directory. This is the profile to add to your home configuration file:

```
; Example 4.2

[bigproject]
command: ${VAR:cmd}
base: ${VAR:home}/Work
backup: /media/antonio/WORKDEV
directories: Big Project
```

At work, the profile would be similar, of course, taking into account the different user and directory names:

```
; Example 5

[bigproject]
command: rsync -aPhv --delete
base: /home/adias/Projects
backup: /media/adias/WORKDEV
directories: Big Project
```

At the end of each day, I would have to upload the changes I made from the work PC to the USB device:

```
$ mybkp -i bigproject
```

and at home, download (*restore*) the changes to my home desktop:

```
$ mybkp -ir bigproject
```

If I do some work at home, I would have to do the same thing (but in the opposite direction) before going to work.

If I had `ssh` access to the work computer, I would only need to use *My Backup* at home:

```
; Example 4.3

[bigproject]
command: ${VAR:cmd}
base: adias@cooljobs.com:/home/adias/Projects
backup: ${VAR:home}/Work
directories: Big Project
```

Getting home after a day at work I would only have to download the changes from the work PC and, after working on the project at home, upload them to the

machine at work:

```
$ mybkp -i bigproject
```

... do some work ...

```
$ mybkp -ir bigproject
```

## Snapshot backups and backup rotation

The last use case in this manual is for those who like to have *snapshot* kind of backups, i.e. backups that reflect the state at differents points in time. This may be useful to revert changes made before the last backup.

The most basic way of doing this with *My Backup* is as explained in the **USAGE** section, using the `--backup_directory` option to indicate the directory where the bakup will reside. If I wanted to keep snapshots of the `Work` and `Projects` directories I would start to make an appropriate profile:

```
; Example 4.4

[projects]
command: ${VAR:cmd}
base: ${VAR:home}
backup: /media/antonio/BKP_DISK/snapshots
directories: Projects, Work
```

Then, for each snapshot I would have to call `mybkp` on that profile and with the directory name I wish to use under the `snapshots` directory of the backup external drive. One possibility is to use the date when the backup is made as the directory name:

```
$ mybkp -i -b 2020-09-03 projects
```

This would copy the `Projects` and `Work` directories to the `snapshots/2020-09-03` directory of the backup disk. I could simplify the calling process by creating an alias that would insert the date automatically, using the `date` command with a custom format string:

```
alias smybkp='mybkp -i -b `date +%Y-%m-%d`'
```

Now I would only have to use

```
$ smybkp projects
```

to achieve the same result.

With this program option it is also possible to create a *circular* bakup, i.e. a series of backups where the last overwrites the most ancient one. Let's say I want to have a 7-deep circular backup, one for each day of the week. Everyday I do a backup, I will overwrite the backup of the the same day of the last week. Take this backup using the `projects` profile on a Monday:

```
$ mybkp -i -b Monday projects
```

The backup directory would be `snapshots/Monday` on the backup disk. I could also simplify it with an alias:

```
alias cmybkp='mybkp -i -b `date +%A`'
```

This completes the uses cases and I think it covers most things you could do with this program. If you have any question, please contact me by email using the address at the top of this manual.

## LICENSE