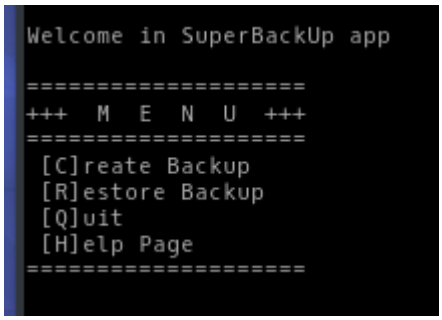
Brief

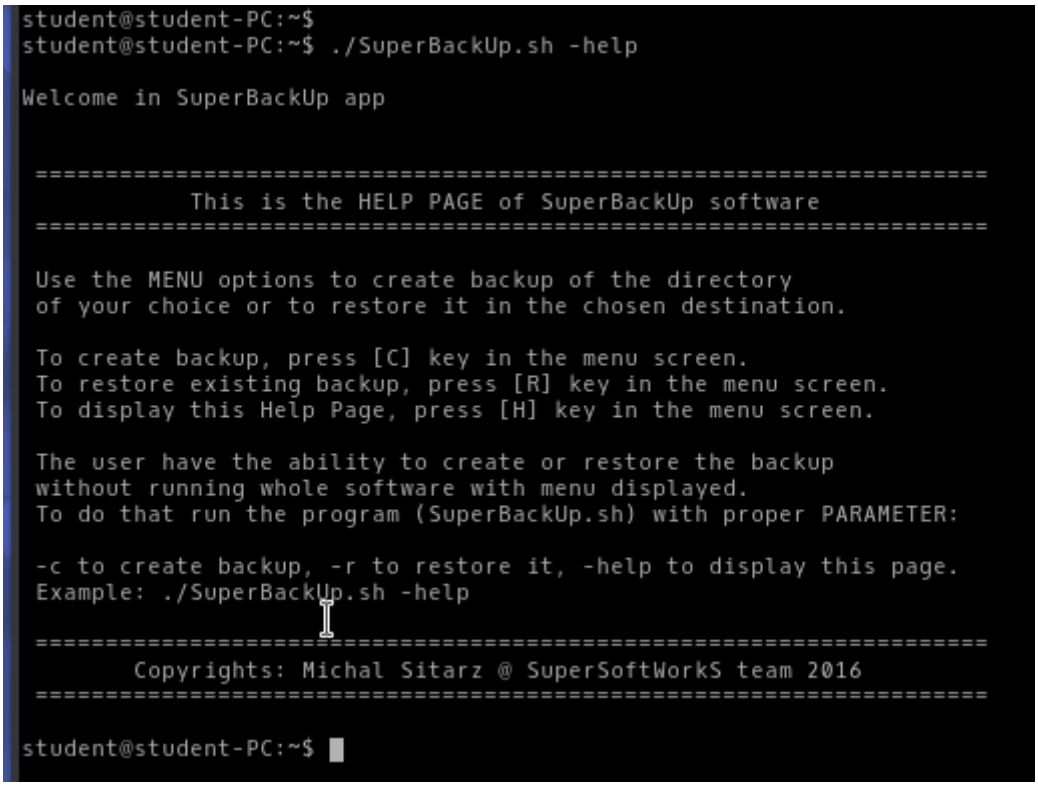
This script is used to create and restore backups of directories.
It is *menu driven* with 4 menu items:

- [C]reate Backup
- [R]estore backup
- [Q]uit
- [H]elp Page



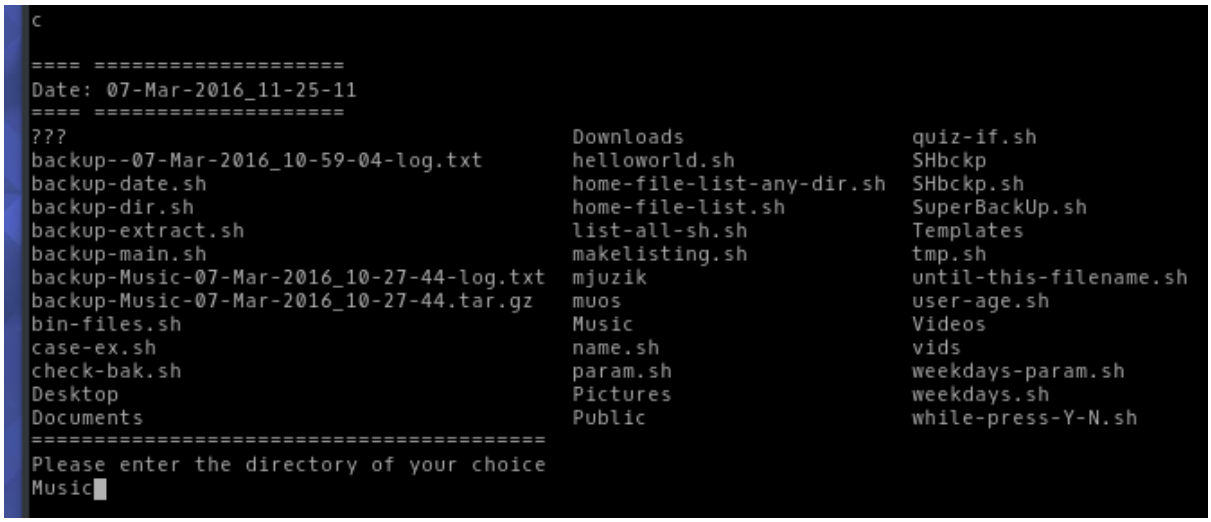
The user is able to choose the right option by pressing the key, highlighted inside brackets [ ].
The user can use either lower case or upper case letters.

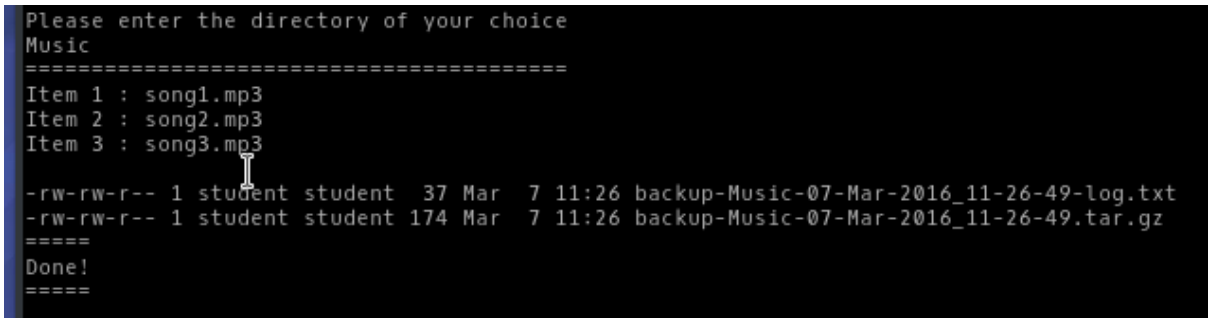
Advanced users are able to use *parameters* to quickly run the common options from command line using parameters: **-c** to create a backup, **-r** to restore a backup and **-help** to display the help page.



Creating a backup

To create the backup of a directory of user's choice
run the **[C]reate Backup** option (or use relevant parameter: **-c**)



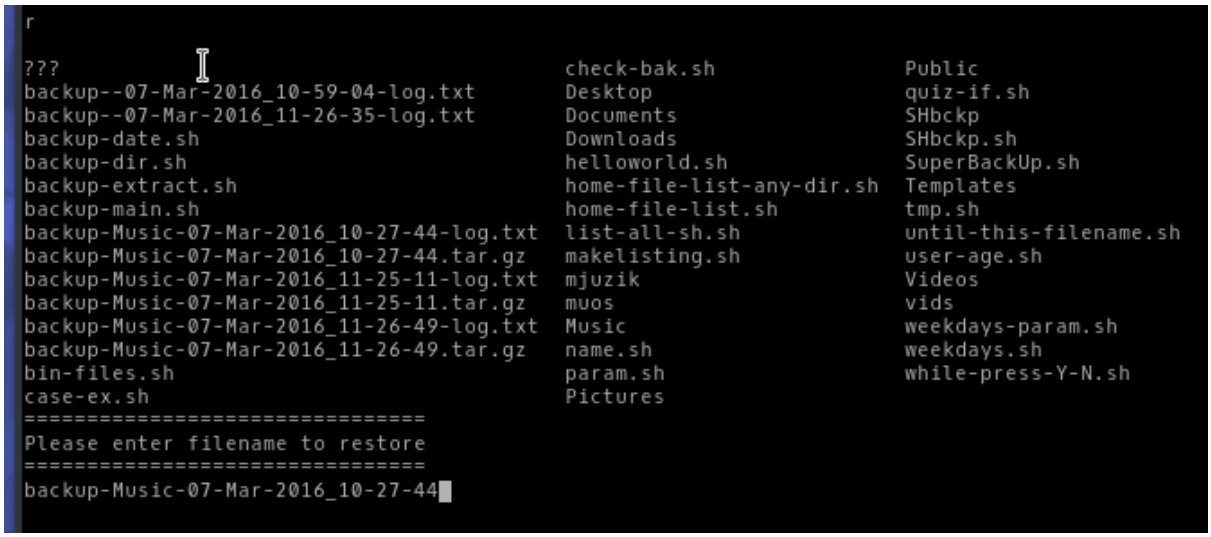


The script will create a backup file (date stamped) with proper extension .tar.gz
The log file (date stamped as well) will be also created with backup file list.

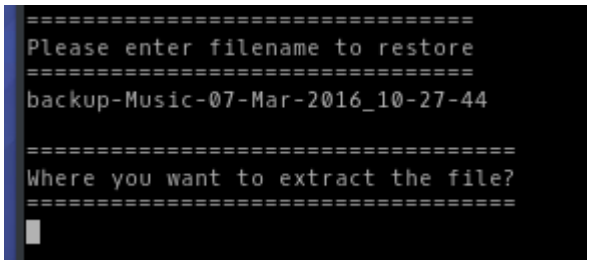
"Done!" message means that the backup has been made properly without any errors.

Restoring a backup

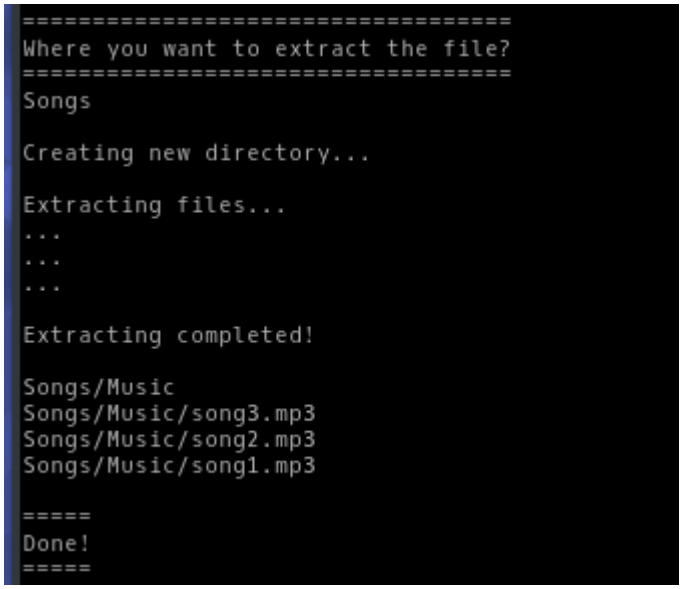
To restore an existing backup of user's choice
run the **[R]estore Backup** option (or use relevant parameter: **-r**)



It is very important to **enter proper filename** (without .tar.gz extension).



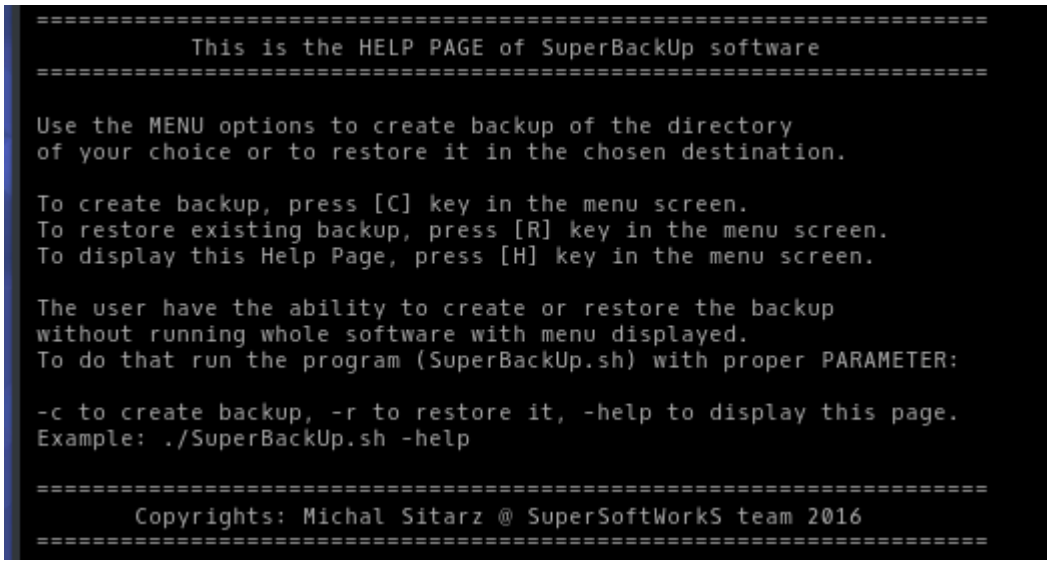
The user can either enter existing target directory or new one.



"Done!" message means that the backup has been restored properly without any errors.

Using Help Page

To use a help page run the **[H]elp Page** option (or use relevant parameter: **-help**)



Test Log

= TEST LOG = Tested file name: SuperBackUp.sh			Tester: Michal Sitarz	Date: 21/03/2016	
Test Case No.	Operation Tested	Input	Expected Output	Actual Output	Correct ?
Testing: menu options					
1	Menu option: [H]elp page	H	Display *Help Page*	Displaying *Help Page*	✓
2	Menu option: [Q]uit	Q	Quit the program	Program finished running.	✓
3	Menu option: [Q]uit	q	Quit the program	Program finished running.	✓
4	Menu option: [C]reate Backup	c	Run *Create Backup* script	Proper script has been run.	✓
5	Menu option: [R]estore Backup	R	Run *Restore Backup* script	Proper script has been run.	✓
6	Menu option: other keys	f	Display *Wrong key pressed* message	Proper message displayed	✓
7	Menu option: other keys	5	Display *Wrong key pressed* message	Proper message displayed	✓
8	Menu option: other keys	exit	Display *Wrong key pressed* message	Proper message displayed	✓
9	Menu option: other keys	only *Enter* key pressed	Display *Wrong key pressed* message	Proper message displayed	✓
Testing: passing the parameter to the program					
10	Passing the parameter	./SuperBackUp.sh -help	Display *Help Page*	Displaying *Help Page*	✓
11	Passing the parameter	./SuperBackUp.sh -c	Run *Create Backup* script	Proper script has been run.	✓
12	Passing the parameter	./SuperBackUp.sh -r	Run *Restore Backup* script	Proper script has been run.	✓
13	Passing the parameter	./SuperBackUp.sh -m	Display message *Wrong parameter has been used*	Run the program without any parameter passed	X
Testing: Create Backup script proper work					
14	Entering the directory name to **create** a backup	Music (existing directory)	Display message confirming completed task (backup created).	Proper message displayed with list of backup archive created and a log file.	✓

15	Entering the directory name to **create** a backup	beef (not existing directory)	Display message about non-existing directory	*beef is not a directory* message displayed	✓
16	Entering the directory name to **create** a backup	5 (not existing directory)	Display message about non-existing directory	*5 is not a directory* message displayed	✓
17	Entering the directory name to **create** a backup	*blank name* (just *Enter* key pressed without any directory name)	Display message about non-existing directory	The script displays an error, but run after, doesn't create a backup file, but lists the content of current directory and displays the *Done!* Message, which is incorrect.	X
Testing: Restore Backup script proper work					
18	Entering the directory name to **restore** a backup	backup-Music-21-Mar-2016_11-09-45	Display message with question: *Where to extract the file?*	*Where to extract the file?* Message displayed.	✓
19	Entering the directory name to **restore** a backup	backup-	Display message about non-existing file to restore	Proper message displayed	✓
20	Entering the directory name to **restore** a backup	555	Display message about non-existing file to restore	Proper message displayed	✓
21	Entering the directory name to **restore** a backup	Docs	Display message about non-existing file to restore	Proper message displayed	✓
22	Entering the directory name to **restore** a backup	backup-Music-21-Mar-2016_11-09-45**.tar.gz**	Display message with question: *Where to extract the file?*	Displays message about non-existing file to restore	X
Testing: Restore Backup script proper work – extracting files					
23	Entering the directory name to **extract** a backup	Music (existing directory)	Display message confirming completed extraction and list of files + confirmation of backup restore	Proper message displayed with list of files restored.	✓
24	Entering the directory name to **extract** a backup	muzzik (not existing directory)	Display message confirming new directory creation, completed extraction and list of files + confirmation of backup restore	All proper messages displayed (including new directory creation) with list of files restored.	✓

Work Log

1st step: Plan the program

When I was planning the program, first thing I've done was to check what exact requirements have to be met. This gave me a broad view of the program and how it should look/work like.
Then I started to plan the program. I sketched the plan of the program on the paper including menu with its options and a brief function of each of them. I also pointed out the use of parameters, which will be passed to the program from command line and will run corresponding options to them, without running whole program. I wrote down all the variables I will need to use (including global ones) and I planned the naming scheme for all of them, as well as for functions I will use.

2nd step: Plan the work

When I knew what I will need to make I had to plan the work that I will need to put into the creation of the program. I knew I had generally working scripts for creating the backup, as well as restoring the backup, written by me during the exercises, but they needed some tweaks and debugging, so I didn't have to write them from the scratches.

I decided that I will start with the fully functional menu, which will give me the skeleton of the program and then I will implement the use of the parameters, to call the same functions that have been included in the menu.

Then I will implement the working scripts of creating the backup and restoring the backup. After that I will solve the errors, tweak the scripts and the whole working program, including all user friendly improvements. Then testing phase and documentation.

3nd step: Write a pseudo code

When I had all planned out I started to write a pseudo code, having a sketch program in mind. During the pseudo code writing I had a chance to rethink the flow of the program. It allowed me to design more precisely the program's main constructs and where to use which. At this stage I also made a choice to use a "case switch", rather than "if then statement" to control the menu options, as it will be easier to implement the parameters passing functionality.

4th step: Write a proper code

At this stage I wrote a proper scripting code starting with the menu, as I planned, following the pseudo code. With working menu I quickly tested it, then I implemented the passing of parameters and again I tested the proper functionality of the program. After that I implemented the *Create Backup* script and *Restore Backup* script, including minor changes in those scripts for full integrity.

5th step: Tweaks and improvements

With fully working program, menu driven, with the ability to pass the parameters to run the corresponding options and with working scripts, it was the right moment to improve the visual side of the program to be more user friendly and to make a proper Help Page (to help future users to use the program easily and effectively). I used special characters inside the code to highlight all the important messages, such as: errors, confirmations, user choices, etc.

6th step: Testing and documentation phase

With ready-to-use program I started the testing phase. Test log (attached above) will show and explain whole process of testing strategy. After testing phase I created the documentation with brief overview of the program, its function and options available. Corresponding screenshots are included, as well as the content of the help page. This documentation is finished by the work log.