




Universal Backup Tool

Team 2

User Manual



UBT is a free cross platform program that can allow users to upload select directories from their computers and create supplementary exact copies.

The services are twofold: automatic and manual backup. (one for uninterrupted, timed background backups and one for when you want to be sure everything is saved!)

Creates a backup directory (source) and copies that over to a destination directory (destination).



It checks whether or not a file exists, it backs it up incrementally



if it already exists, it check if its the newest and will not copy



if it doesn't exist, it will create it



it will always check files added, it will check if the files in the source exist in the destination directory

Welcome to the Universal backup tool. You can quickly back up personal files with just a few clicks. Your personal files include all non-system files on Windows/Linux/Mac.

Features:

- Seamless GUI
- Bash powered:
Linux,
windows, and
Mac
compatibility
assured
- Incremental

Requirements:

- Conventional operating system
- Updated software drivers

Scrum board



	To Do	In Progress	Testing	Done
RSYNC	-Removal of redundant files	-Additional Scripting	-Shows Time Stamp	-Copies Files
CRONTAB	-Give users the option of which incremental backup time slot	-Syntax of scheduling	-Running the same time every month,day, or hour	-Base scripting
GUI	-When the other functions are implemented, create echoes for them.	-Buttons	-Executing	-Echoes -Dimensions
SCRIPTING	-Solve error(s)	-Additional scripting as needed	-Copying files	-Base scripting

RSYNC (Abolaji)

- Copies all files and folders including everything the subdirectories to a USB flash drive named DocsBackup.
- Creates and shows original timestamps. Creation and modification dates for files are set to current time.
- Archive mode maintains time stamps, copies recursively, and keeps all files
- Removes any directory or file on the destination directory that is not present on the source.
- Backs up in the form of Year, Month, and Day.

```
9 Rsync
...  @@ -0,0 +1,9 @@
1  + rsync -r /Users/skibo/ /Desktop /Volumes/DocsBackup
2  +
3  + rsync -r -t /Users/skibo/Desktop /Volumes/DocsBackup
4  +
5  + rsync -a /Users/skibo/Documents /Volumes/DocsBackup
6  +
7  + rsync -rt -delete /Users/skibo/Desktop /Volumes/DocsBackup
8  +
9  + rsync -rtb -backup-dir="backup ${date +%Y-%m-%d}" -delete /Users/skibo/Desktop /Volumes/DocsBackup
```



RSYNC (Abolaji)

- This gives more feedback on what rsync is doing
- Excludes few directories not needed

```
SKIBOs-MBP:~ skibo$ rsync -a --progress /Users/skibo/Documents/ /Volumes/Docsba  
ckup/Documents
```

```
SKIBOs-MBP:~ skibo$ rsync -a --exclude Parallels/Users/skibo/~ /Documents/ /Volu  
mes/backup/Documents
```



GUI (Hernan)

- This function is to create a popup textbox that will ask for the files backup.
- There's 2 buttons to this function, both being the yes and no button.
- The first line imports the tool that is used for the GUI, then the loop is defined.
- In the end, this will ask if you want to commence the backup, and you can select either yes or no.

```
from tkinter import *
backup=Tk()
backup.title("ALERT")
backup.geometry("300x250")

Label(backup, text="Do you want to backup?").pack()
header= Entry(backup, textvariable="Do you want to backup?")

Button(backup, text="Yes", width=10, height=1).pack()

Button(backup, text="No", width=10, height=1).pack()
backup.mainloop()
```



GUI (Hernan)

- In the first 2 lines we are importing the date and time.

This is because we want the backup to have the specific time and date from when it was done.
- The line where “def” starts, it begins with the function to backup files.
- In the “src_file” line, this is the source file from where we are backing up from
- The “bkp_file_name” will be our modified backup file of the

source file. The “shutil” creates the backup and then says backup finished, when the backup is completed.

```
from datetime import date
import shutil

def take_bkp(src_file_name, bkp_file_name = None,
             src_file_loc = '', bkp_file_loc = ''):

    src_file = src_file_loc+src_file_name

    src_file= src_file_loc+src_file_name

    bkp_file = bkp_file_loc+bkp_file_name

    today = date.today()
    date_format = today.strftime("%d_%b_%Y_")

    if bkp_file_name is None or not bkp_file_name:
        bkp_file_name = src_file_name
        bkp_file = bkp_file_loc+date_format+bkp_file_name

    else:
        bkp_file = bkp_file_loc+date_format+bkp_file_name

    shutil.copy2(src_file, bkp_file)

    print("Backup Finished")
```



GUI (Hernan)

- The first three lines we are just defining locations for the backups. We also define that the max backup amount stored will be 5.
- The “assert” and “mkdir” lines are making sure that these directories for backup location exist, If not one will be created.
- The “existing backups” is a line that we use to get Those backups that have already been created and stored.
- While the “oldest to newest” line is making sure that there is no more backups past 5, and if there is then the oldest backup will be deleted for the new one.

```
OBJECT_TO_BACKUP = '/home/server/data.db'  
BACKUP_DIRECTORY = '/home/server/backup'  
MAX_BACKUP_AMOUNT = 5
```

```
object_to_backup_path = Path(OBJECT_TO_BACKUP)  
backup_directory_path = Path(BACKUP_DIRECTORY)  
assert object_to_backup_path.exists()
```

```
backup_directory_path.mkdir(parents=True, exists_ok=True)
```

```
existing_backups = [  
    x for x in backup_directory_path.iterdir()  
    if x.is_file() and x.name.startswith('backup-')  
]
```

```
oldest_to_newest_backup_by_name = list(sorted(existing_backups, key=lambda f: f.name))
```

```
while len(oldest_to_newest_backup_by_name) >= MAX_BACKUP_AMOUNT:  
    backup_to_delete = oldest_to_newest_backup_by_name.pop(0)  
    backup_to_delete.unlink()
```




CRONTAB (Jerry)

- Beginning CronTab to schedule when we want the backup tool to actually backup the files that we want.
- Example shows the 1st of every month at 12pm
- 00 1 * * = month, hour, day of month, month and day of week

```
1  +# m h dom mon dow  command
2  +0 0 1 * * bash /usr/local/bin/backup.sh
3  +Automatic Backup every 1st of every month at 12P.M
4  +m=minute
5  +h= hour
6  +dom=day of the month
7  +mon=month
8  +dow=day of the week
```



SCRIPTING (Jerry)

- The initial start of the tool using Bash
- Backing up the directory to the destination external disk
- A popup window notifying the user that the backup will start
- Coding the variables in Bash
- Destination Directory
- Exclusion of files
- The server, user and options

```
#!/bin/bash
set -x
clear
#Program will backup the directory to a external disk automatically
set -x
Osascript -e 'display notification "Backup is about to begin"'
rm ./log
SOURCE=$1
DEST_DIRECTORY=$2
EXCLUDE_FILE=$3
SSH_SERVER=$4
SSH_USER=$5
SSH_OPTION=$6
+
SSH_REMOTE_SERVER=$5@$4
SSH=ssh $6 $SSH_REMOTE_SERVER
SSH_DEST_DIRECTORY=
+
PWD=$(pwd)
ID=$(id)
echo $ID is running this script from $PWD
```



SCRIPTING (Jerry)

- Step by Step of the Backups function
- 1st checking external disk
- 2nd checking that the directory is there as well as backup
- 3rd complete the latest backup of directory
- Backup the directory for the current date
- Exclude any files that user does not want backed up

```
+#First - check that external disk is connected
+echo "Source Directory: $1"
+echo "BACKDIR: $BACKUPDIR"
+
+#Second - if the directory exists, check if the "backup" directory exists
+if ($SSH '[ ! -d $BACKUPDIR directory does not exist. Creating."
+mkdir $BACKUPDIR
+fi
+
+#Third - get the latest backup dir
+LATEST=$(SSH 'ls' '-l' '@eaDir' '-r' $DIR ' | ' 'head' '-n1')
+echo "LATEST: $LATEST"
+
+DATE='date
+TARGET=$BACKUPDIR/$DATE
+echo "TARGET: $TARGET"
+
+EXCLUDE_FILE=$3
+echo "EXCLUDE_FILE: $EXCLUDE_FILE"
+#take relative path
+COMPLATE_LINK_DEST_DIR="../$LATEST"
```



SCRIPTING (Jerry)

- Linking the old directory to the new destination directory
- An example of a users local disk location with rsync
- Excluding the files the user does not want and targeting the destination of the backup

```
+echo "linking old $COMPLETE_LINK_DEST_DIR to $TARGET..."
+
+/usr/local/bin/rsync -av --numeric-ids --progress --delete -e "ssh -p 2222" --exclude-from="$EXCLUDE_FILE" --link-dest="$COMPLETE_LINK_DEST_DIR" $1 $TARGET | tee $HOME/Tools/Scripts/BackupScript/backup$DATE.log
+/usr/local/bin/rsync -av --numeric-ids --progress --delete -e "ssh -p 2222" --exclude-from="$EXCLUDE_FILE" --link-dest="$COMPLETE_LINK_DEST_DIR" $1 $TARGET
+| tee $HOME/Tools/Scripts/BackupScript/backup$DATE.log
```

Scripting (Mykael)

My variables to read and copy files to and from

(For loop - checks the files, tells it to look in the source and print format)

(If statement: (-a checks if destination file exists) this is looping all the files in the source and listing everything in a recursive manner. meaning all the files underneath source are in a relative path, so it basically checks anything behind source and if it exists behind destination. So it checks if the copied files have already been created. Then if its there, it doesn't need to be copied)

```
40
41 source=/Users/micha/OneDrive/Desktop/backup/source
42 dest=/Users/micha/OneDrive/Desktop/backuo/destination
43
44 for file in $(find $source -printf "%P\n") ; do
45     if [ -a $dest/$file ] ; then
46         if [ $source/$file -nt $dest/$file ] ; then
47             echo "Copying: newer file"
48             cp -r $source/$file $dest/$file
49         else
50             echo "File $file already exists, skipping."
51         fi
52     else
53         echo "File is being copied into $dest"
54         cp -r $source/$file $dest/$file
55     fi
56 done
57
58
```

if [\$source/\$file -nt \$dest/\$file] ; then	(if the source file is newer than the destination)
echo "Copying: newer file"	(it will leave a message)
cp -r \$source/\$file \$dest/\$file	(it copies recursively, so that it also copies all of the underlying folders and stuff)
else	(if thats not the case)
echo "File \$file already exists"	(report that it would be redundant, and basically skip copying it)
fi	(end it with fi)
else	(if that's not the case then)
echo "\$file is being copied into \$dest"	(report the file being copied)
cp -r \$source/\$file \$dest/\$file	
fi	(end with fi)
done	(done)

