

# MOBI Backup (rsync wrapper)

like probably every Linux admin, I eventually came to the point where I felt like it was time to write **My Own Backup Implementation** as an rsync wrapper to do some backups. This script is in its functionality very similar to what [rubi](#) does: it creates a new sub-directory with the date of the backup as directory each time the backup is run. every backup directory contains a full backup of the source, but only the difference since the last backup does actually need to be synced. when i say the difference i mean files that have changed.. yes, files, not blocks! so if your 2GB log file gets a new line, 2GB will have to be downloaded. but if your file does not change, it will be hard-linked to the previous backup and therefore nothing needs to be downloaded.

to achieve this, I use rsync's `--link-dst` option.

In most cases, this proves to be simple but still efficient enough, rather than trying block-level incrementals.

at the end of a successful backup, a rotation is made and old backups are being deleted where appropriate. also a summary email is sent to the admin.

so here is the script.. use it at your own risk and let me know if you find bugs or have contributions to make. simply send me an email to contact at [psuter dot ch](#).

to configure, simply edit the lines or add more blocks after the

```
#####  
### Backup Job Definitions start here #####  
#####
```

comment in the script. some lines above the comment you can find different config options mixed with some code that should not be changed ;).. i know.. usability and such wasn't the main focus here but instead i wanted to keep everything in a file and as simple as possible to read the code and modify it to your own needs.

the script will write a hidden file named `.lastdst` to the backup base directory for each backup job. this file always contains the folder name of the sub directory of the last successful backup.

[mobi.sh](#)

```
#!/bin/bash  
# (c) Pascal Suter, DALCO AG, 2018. Use at your own risk  
# no use without prior permission from DALCO AG  
# pascal.suter@dalco.ch  
# the latest version and some documentation for this script can be  
# found on http://wiki.psuter.ch/doku.php?id=mobi\_backup  
# version 1.1  
# replaced ps aux | grep rsync style locking with flock locking to  
# allow this script to run on servers that use rsync for other stuff as  
# well :)  
# version 1.2  
# added eval in front of rsync call to properly evaluate the $OPTIONS
```

```

variable contents when running the command
# version 1.3
# moved log from rsyncs stderr to a separate .err file which makes
finding the relevant error messages in the rsync output alot easier

report() {
    LOG=$1
    error=0
    # get all jobs that where started
    jobs=`grep "^Backup of" $LOG | awk '{print $3}'`
    # get all jobs that where successfully completed
    successful_jobs=`grep "^Backup for .* completed successfully" $LOG
| awk '{print $3,"finished on",$7,$8,$9,$10,$11}'`
    # get all jobs that did not complete
    failed_jobs=`grep "^Backup failed" $LOG | awk '{print $4,"stopped
on", $15,$16,$17,$18,$19,$20}' | tr -d ",() "`
    # get remainig jobs without status report
    remaining_jobs="$jobs"
    for job in `echo -e "$successful_jobs" | awk '{print $1}'; do
        remaining_jobs=`echo -e "$remaining_jobs" | sed -e "s/$job$/"`
    done
    for job in `echo -e "$failed_jobs" | awk '{print $1}'; do
        remaining_jobs=`echo -e "$remaining_jobs" | sed -e "s/$job$/"`
    done
    remaining_jobs=`echo -e "$remaining_jobs" | sed -e '/^$/d'`

    # write a report summary
    echo "#####"
    echo "Backup report for ${1}"
    echo "#####"
    if [ "$remaining_jobs" != "" ]; then
        error=1
        echo "===== "
        echo "== Jobs in an unknown state (still running?) == "
        echo "===== "
        echo -e "$remaining_jobs"
    fi
    if [ "$failed_jobs" != "" ]; then
        error=1
        echo "===== "
        echo "== Failed Jobs ===== "
        echo "===== "
        echo -e "$failed_jobs"
    fi
    echo "===== "
    echo "== Jobs successfully completed ===== "
    echo "===== "
    echo -e "$successful_jobs"
    echo "===== "
    echo "== Jobs started ===== "
    echo "===== "

```

```
echo -e "$jobs"
if [ $error -gt 0 ]; then
    SUBJECT="FAIL"
else
    SUBJECT="SUCCESS"
fi
}

rotate() {
    # remove old backups and keep only a certain amount of consecutive
    # and monthly backups
    # parameters:
    # arg 1: number of successful consecutive backups to keep (max. 1
    # per day will be kept,
    #         if there is more than one backup per day, the newest will
    # be kept, the rest will be deleted
    # arg 2: number of successful monthly backups to keep
    #         keeps last backup of the month starting on the month before
    # the oldest
    #         of the consecutive backups that have been kept back
    # arg 3: directory

    olddir=`pwd`
    numConsec=$1
    numMonthly=$2
    dir="$3"
    if [ $numConsec -lt 1 ]; then
        echo "first argument of rotate() should be number of
consecutive backups to keep. number given was smaller than 1, this must
be a mistake!"
        exit 1
    fi

    if [ ! -d "$dir" ]; then
        echo "the third argument of rotate() should be the backup
directory to clean up. the given directory does not exist"
        exit 1
    fi
    cd "$dir"
    echo "Starting Cleanup Process for `pwd`"

    # get all successful backups
    backups=`grep -l "completed successfully" *.log | sort | sed -e
's/\.log$//`

    # keep the last $numConsec consecutive backups
    keep=`echo "$backups" | awk -F - '{print $1}' | grep -v -P "^s*$"
| uniq | tail -n $numConsec`

    # check if we even have more than $numConsec backups yet:
    if [ `echo "$backups" | wc -l` -lt $numConsec ]; then
```

```

    echo "we do not have enough backups to start deleting yet"
    exit 0
fi

# get the oldest of the last $numConsec backups:
lastdate=`echo "$keep" | head -n 1`
lastyear=`echo $lastdate | awk -F . '{print $1}'`
lastmonth=`echo $lastdate | awk -F . '{print $2}'`
lastday=`echo $lastdate | awk -F . '{print $3}'`

# calculate the last $numMonthly months to keep:
month=$lastmonth
year=$lastyear
for i in `seq 1 $numMonthly`; do
    month=`expr $month + 0`
    let month--
    if [ $month -lt 1 ]; then
        month=12
        let year--
    fi
    month=`printf "%02d\n" $month`
    keep=`echo -e "$keep\n$year.$month"`
done
keepdates=""
for i in $keep ; do
    latest=`echo "$backups" | grep "^$i" | tail -n 1`
    keepdates=`echo -e "$keepdates\n$latest"`
done

keepdates=`echo "$keepdates" | grep -v -P "^s*$"`

delete=`ls *.log | sed -e 's/\.log$//' | sort | uniq`
delbackups=$backups
for i in $keepdates; do
    delete=`echo "$delete" | grep -v "$i"`
    delbackups=`echo "$delbackups" | grep -v "$i"`
done

delbackups=`echo "$delbackups" | grep -v -P "^s*$"`

echo "All Backups:"
echo "$backups"
echo "====="
echo "Backups to delete:"
echo "$delbackups"
echo "====="
echo "Backups to keep:"
echo "$keepdates"

```

*#sanity check before deleting backups: check if enough backups will be left after deleting everything else*

```

numBD=`echo "$delbackups" | wc -l`
numBT=`echo "$backups" | wc -l`
survivors=`expr $numBT - $numBD`
if [ $survivors -lt $numConsec ]; then
    echo "ERROR: somehow too many backups would have been deletd,
this should not happen!, aborting"
    exit 1
else
    echo "$survivors backups will be left after deleting obsolete
backups"
fi

echo "$delete" | xargs -I DDD /bin/bash -c 'echo "deleting DDD*";
rm -rf --one-file-system DDD*'
echo "cleanup complete for `pwd`"
}

run() {
    echo "Backup of $BACKUPNAME was queud on Position $INTPROCID";
    keepWaiting=1
    while [ $keepWaiting -gt 0 ]; do
        keepWaiting=0
        #check if enough processes that where launched before me have
finished for me to start my work
        for ((i=1; i<=$(expr $INTPROCID - $PARALLELPROCS); i++)); do
            if [ `grep -c " $i " /dev/shm/backup_finished` -eq 0 ];
then
                keepWaiting=1
            fi
        done
        #echo "waiting patiently to start the backup of $BACKUPNAME
with my ID $INTPROCID"
        sleep 5
    done

    processes=`lsof ${LOCKDIR}/* 2>/dev/null | grep -v flock | grep
backup | awk -F / '{print $NF}' | sort | uniq | wc -l`

    while [ $processes -gt $PARALLELPROCS ]; do
        echo "Waiting for another rsync to complete before I can start
with ${BACKUPNAME}."
        sleep 10
        processes=`lsof ${LOCKDIR}/* 2>/dev/null | grep -v flock | grep
backup | awk -F / '{print $NF}' | sort | uniq | wc -l`
    done

    echo "Starting Backup for ${BACKUPNAME} at `date`" | tee -a
${BASEDST}/${DSTDIR}.log

    mkdir -p ${BASEDST}

```

```

# read the .lastdst file and check if it is either empty (full
backup) or if it contains a valid directory (incremental backup).
#if it is not empty and the content is not the name of a directory,
the backup will be aborted

```

```

OLDDST="`cat $BASEDST/.lastdst 2>/dev/null`"

go=1
if [ -n "$OLDDST" ]; then
    if [ ! -d "${BASEDST}/${OLDDST}" ]; then
        echo "the given last destination $OLDDST does not exist,
will not proceed with the backup in order to not accidentally do a full
backup"
        go=0
    fi
fi

if [ $go -eq 1 ]; then
    echo "flock -E 66 -n ${LOCKDIR}/${BACKUPNAME} rsync -aAHXv
${OPTIONS} --relative --delete --numeric-ids --bwlimit=${BWLIMIT} --
progress --link-dest="../${OLDDST}" --one-file-system ${SOURCE}
${BASEDST}/current >> ${BASEDST}/${DSTDIR}.log
2>${BASEDST}/${DSTDIR}.err"
    eval flock -E 66 -n ${LOCKDIR}/${BACKUPNAME} rsync -aAHXv
${OPTIONS} --relative --delete --numeric-ids --bwlimit=${BWLIMIT} --
progress --link-dest="../${OLDDST}" --one-file-system ${SOURCE}
${BASEDST}/current >> ${BASEDST}/${DSTDIR}.log
2>${BASEDST}/${DSTDIR}.err
    ret=$?
else
    ret=1
fi

if [ $ret -eq 0 -o $ret -eq 24 ]; then
    mv ${BASEDST}/current ${BASEDST}/${DSTDIR}
    echo -n ${DSTDIR} > ${BASEDST}/.lastdst
    if [ $ret -eq 0 ]; then
        echo "Backup for ${BACKUPNAME} completed successfully at
`date`" | tee -a ${BASEDST}/${DSTDIR}.log
    else
        echo "Backup for ${BACKUPNAME} completed successfully at
`date` but some files vanished before they could be copied" | tee -a
${BASEDST}/${DSTDIR}.log
    fi
    echo "rotating old backups" | tee -a ${BASEDST}/${DSTDIR}.log
    rotate "$KEEPCC" "$KEEPMP" "$BASEDST" >> ${BASEDST}/${DSTDIR}.log
    echo "rotation completed at `date`" | tee -a
${BASEDST}/${DSTDIR}.log
elif [ $ret -eq 66 ]; then
    echo "there are other rsync jobs running for this host,

```

```

skipping backup this time" | tee -a ${BASEDST}/${DSTDIR}.log
    echo -n "$INTPROCID " >> /dev/shm/backup_finished
    exit 1;
else
    echo "Backup failed for ${BACKUPNAME} with errorcode ${ret},
keeping current progress to continue next time (`date`). to debug check
${BASEDST}/${DSTDIR}.err" | tee -a ${BASEDST}/${DSTDIR}.log
fi

    echo -n "$INTPROCID " >> /dev/shm/backup_finished
    exit
}

start(){
    mkdir -p ${LOCKDIR} 2>/dev/null
    INTPROCID=`expr $INTPROCID + 1`
    echo "added backup for $BACKUPNAME to the que on position
$INTPROCID" | tee -a $MASTERLOG
    run | tee -a $MASTERLOG &
}

# if "rotate" is given as first argument, simply run a rotation in the
current working directory with default values
if [ "$1" == "rotate" ]; then
    rotate 30 12 "`pwd`"
    exit;
fi

# if "report" is given as first argument, generate a report out of the
master log file that was passed as second argument and exit
if [ "$1" == "report" ]; then
    if [ ! -f "$2" ]; then
        echo "log file not found. please provide the full path of the
log file as second argument"
        exit 1
    fi
    report $2
    exit
fi

DSTDIR=`date +%Y.%m.%d-%H%M` # always start with `date +%Y.%m.%d... as
this is needed for rotation to work later on!

MASTERLOG="/tmp/`date +%Y.%m.%d-%H%M`.log"
REPORT_RECIPIENTS="root" # separate multiple recipients with space
REPORT_SUBJECT="Backup Report for `hostname`"

#check if this script is not still running in an old backup
SELF=`basename "$0"`

if [ `ps aux | grep $SELF | grep -v '/bin/sh' | grep -vc grep` -gt 2 ];

```

```

then
    echo "another backup is still running, let it finish first" | tee -
a $MASTERLOG
    echo "`ps aux | grep $SELF | grep -v '/bin/sh' | grep -v grep` " |
tee -a $MASTERLOG
    exit 1;
fi

echo -n " " > /dev/shm/backup_finished
INTPROCID=0
PARALLELPROCS=8                # how many processes (rsync jobs)
should be started in parallel? it makes only sense to set this variable
once
LOCKDIR=/var/lock/backup        # directory to keep the lock files
which are used to prevent parallel execution of backups on the same
host

#####
### Backup Job Definitions start here #####
#####

BACKUPNAME="myself"            # this name will be used in status
reports, log files and in the backup path, this does not need to be the
hostname to connect to!
SOURCE="localhost:/ localhost:/data"    # source paths (space
separated if multiple paths are to be backed up)
BASEDST="/backup"              # base backup dir
KEEPC=30                        # number of successful
consecutive backups to keep
KEEPM=12                        # number of monthly backups to keep starting
after the oldest consecutive backup
OPTIONS=""                     # additional rsync options like for example
"-e /usr/bin/rsh"
start

BACKUPNAME="remote.server.ch"
SOURCE="root@remote.server.ch:/ root@remote.server.ch:/boot/efi
root@remote.server.ch:/data"    # source paths (space separated
if multiple paths are to be backed up)
BASEDST="/remoteBackups/${BACKUPNAME}" # base backup dir
KEEPC=30                        # number of successful
consecutive backups to keep
KEEPM=12                        # number of monthly backups to keep starting
after the oldest consecutive backup
OPTIONS=""                     # additional rsync options like for
example "-e /usr/bin/rsh"
start

#####
### Backup Job Definitions end here #####

```



```
#####

# wait for all sub processes to finish before the main process can
expire, this allows proper killing of all backups
children=`ps aux | grep "$SELF" | grep -v '/bin/sh' | grep -vc "grep"`
while [ $children -gt 2 ]; do
    sleep 1
    hosts=`lsof ${LOCKDIR}/* 2>/dev/null | grep -v flock | grep backup
| awk -F / '{print $NF}' | sort | uniq | tr "\n" " "`
    echo "still have $children processes, currently backing up $hosts"
    children=`ps aux | grep "$SELF" | grep -v '/bin/sh' | grep -vc
"grep"`
done
echo "creating backup Report" | tee -a $MASTERLOG
report $MASTERLOG > /tmp/backupReport.txt
cat /tmp/backupReport.txt | mail -s "$SUBJECT: $REPORT_SUBJECT"
"$REPORT_RECIPIENTS"
rm -f /tmp/backupReport.txt
```

## Migration from RUBI

Since i was using rubi before on many systems (private and customer systems) I'll provide a quick migration guide for those who want to migrate from rubi to mobi:

1. download the above script and save it to /opt/backup/backup.sh
2. edit the script
  1. adjust the REPORT\_RECIPIENTS variable as needed
  2. under the Backup Job Definitions start here write a new block for every backup.
    1. BACKUPNAME can be set as you like. it will be mentioned in the backup report. Usually coming from RUBI this will be set to the same value as SRCHOST was in RUBI
    2. SOURCE needs to list all directories like SRCPART did, but they need to be prepended by the hostname, so SRCPART=/ /boot with SRCHOST=mysrv would be migrated into SOURCE=mysrv:/ mysrv:/boot
    3. BASEDST can be copy/pasted from rubi 1:1
    4. KEEPC could be set to the same value as KEEPDP
3. to go the backup directory for each host and run this command

```
basename $(<lastdst) > .lastdst
```

4. cleanup old log files and lastdst file and whatever else there might be laying around in this folder.
5. remmber to come back and delete the old backups when it's time. old RUBI backups won't be rotated using mobi, this needs to be done manually. mobi will only include backups into the rotation for which it finds a log file of a successful backup job.

From:

<http://wiki.psuter.ch/> - **pswiki**

Permanent link:

[http://wiki.psuter.ch/doku.php?id=mobi\\_backup&rev=1537957708](http://wiki.psuter.ch/doku.php?id=mobi_backup&rev=1537957708)

Last update: **26.09.2018 12:28**

