DAR differential backup mini-howto -EN-

Author:	Grzegorz Adam Hankiewicz
Contact:	gradha@titanium.sabren.com
Date:	2006-10-07 19:22:40 +0200
Version:	H5 (424)
Web site:	http://gradha.sdf-eu.org/textos/backup.en.html
Copyright:	This document has been placed in the public domain.
Translations:	From the web site you can get this document in English, Italian and Spanish.

Contents

Introduction

We all should make backups of our important data. This omnipresent advice is usually ignored by most people. I ignored it too, until I lost a good deal of important data. Not happy enough, I managed to continue loosing data in a few posterior incidents, until I decided that it was enough. Then I browsed Freshmeat for backup solutions allowing differential backup and found DAR.

A complete backup means that all the files falling under your backup policy will be saved. A differential or incremental backup will contain only the files whose contents have changed since the previous backup, either full or differential.

DAR allows you to create easily a set of differential backups. The solution I've developed helps me have an automatic backup solution which runs every night. The first day of the month, a full backup is made. The rest of the month, only differential backups are made. In my situation, very few files change from day to day, sometimes the source code of the project I'm hacking on, and always my mailboxes.

The result is that I can restore the contents of my computer to a specific day with ease, if I ever need to. DAR is a command line program, and it can get slightly complex with a few options. This little mini-howto will explain my custom solution, which is very crude, but works fine for me. Yes, I've actually tested restoring the data from the backup. In fact, during the end of the year 2003 I moved to another country and I took just one CD ROM with me plus a bootable Knoppix, and I recovered the exact state of my Debian installation in a few hours. No customizing, no long installations, no missing files.

This document was written using version 1.3.0 of DAR. When I updated to DAR 2.0.3, everything kept working, I didn't even have to update my backup archives. So it looks like the interface and backup format are pretty stable, or at least backwards compatible. However, don't take everything said here for granted. Verify that the version of DAR you have installed works as expected and you can restore from the generated backup before you have to rely on it.

This version of the text uses reStructuredText (that's what the weird markup in the text version is for). See http://docutils.sourceforge.net/ for more information.

Simple DAR usage

DAR is very similar to tar in the number of options it has: there's plenty for every need, but way too much for beginners to handle. As usual, you can always get help from the program typing `dar -h` or `man dar` after you have installed it. Like tar, there's a set of mandatory switches which define the type of operation you are doing (create, extract, list, etc), and a set of switches which affect the selected option. Just for the sake of it, imagine that you want to backup one folder of your home directory. You would write something like this:

```
dar -c backup_file_without_extension -g file1 -g file2 ... -g fileN
```

The output should be similar to the following:

```
$ dar -c my_backup_file -g safecopy.py/ -g translate_chars.py/


 ----------------------------------------
 15 inode(s) saved
 with 0 hard link(s) recorded
 0 inode(s) not saved (no file change)
 0 inode(s) failed to save (filesystem error)
 4 files(s) ignored (excluded by filters)
 0 files(s) recorded as deleted from reference backup
 ----------------------------------------
 Total number of file considered: 19
$ ls
mailbox_date_trimmer/  my_backup_file.1.dar  sdb.py/
mailbox_reader/        safecopy.py/          translate_chars.py/
```

As you will notice, DAR will add a number and extension to your name. The purpose of the extension is clear, it helps to know visually that the file is a DAR backup. The number is called a *slice*, and this is related to DAR's built-in feature of splitting a backup over several media. If for example you wanted to make a backup to CD ROM, but your directories are bigger than the capacity of one CD ROM, you can tell DAR to split the archive across as many files as needed, which you can later burn to several units.

Would you like to recover that backup? Pretty easy, type the following:

```
$ mkdir temp
$ cd temp
$ dar -x ../my_backup_file
file ownership will not be restored as dar is not run as root.
to avoid this message use -O option [return = OK | esc = cancel]
Continuing...


 ----------------------------------------------
 15 file(s) restored
 0 file(s) not restored (not saved in archive)
 0 file(s) ignored (excluded by filters)
 0 file(s) less recent than the one on filesystem
 0 file(s) failed to restore (filesystem error)
 0 file(s) deleted
 ----------------------------------------------
 Total number of file considered: 15
$ ls
safecopy.py/  translate_chars.py/
```

The backup strategy

The first step to create a good backup is to determine what parts of your system need one. This doesn't necessarily mean that you can't create a full backup, but most likely splitting it in at least two parts is going to help DAR (or any backup tool) a lot.

My home system consists of two hard disks. The first hard disk is split into a 3.8 GB partition where my complete system lives, and another partition of 11 GB where all my music and other temporary files are stored, like a local Debian package repository I make for myself. The second hard disk has a 9.4 GB partition and its only purpose is to serve as backup of the primary disk. I have no interest in backing up my music, because I have all the original CDs lying around and have scripts to re-ogg them.

From the 3.8 GB I want to backup, usually between 1.3 and 1.5 GB are always empty. I will split logically the used 2.3 GB into *system* and *home directories* (at the moment of writing this my home is 588 MB). The reason for this split is that as a normal user, I can only change my home directory and other files from the partitions I won't be backing up. Meanwhile the *system* part of the partition remains pretty stable and unmodified because I rarely (un)install software. In fact, from my *home* directory the only things changing usually will be my `Mail` folder and `projects`, where I put documents like this one and other software I write/hack.

The basic distinction between *home directories* and *system* can be useful in organizations too. If you work for a university, usually all machines will have the same system configuration but depending on the machine their homes will have different data. You can make a

system backup of a single machine, and *home backups* of each computer. Another common configuration is having a centralized server which exports home directories with NFS. Here you only have to backup the server. If you have users with high privileges, leave them the task of doing the *system backup* of their own machines, the exported home is something they can ignore because it will be done at the server machine.

Once you've decided what to backup, you want to decide how to configure DAR for the backups. You can use switches or configuration files. Switches are OK when you don't have many options. Configuration files are better when you want to make different complex inclusion/exclusion rules of what files you want to backup, and more importantly, you can use comments to document the switch, stating for example the reason why you included this or that directory. This can be useful if you come back several months later and you wonder why all those options are there.

For my setup, I'll be running the DAR commands inside shell scripts called periodically by cron (Setting up some scripts to automate the process), so I don't mind having long command lines, and this very same document serves for the purpose of documenting the scripts. If you prefer configuration files, read DAR's documentation to find out how to use them and the format they use.

Making a full backup with DAR

Here is the full command line I'll be using for my *system* backup, running as **root**. Don't worry about the high number of switches, I'll go on describing the purpose of each of them:

```
dar -m 256 -y -s 600M -D -R / -c ‘date -I‘_data -Z "*.gz" \
   -Z "*.bz2" -Z "*.zip" -Z "*.png" -P home/gradha -P tmp \
   -P mnt -P dev/pts -P proc -P floppy -P burner -P cdrom
```

- `-m 256` DAR can compress your backup. The compression is applied to individual files, and it can be bad for small files. By default files with 100 bytes or less won't be compressed. With the `-m` switch I increase this to 256, which seems to work better for all those little configuration files lying under `/etc/` and `/home`. As you see this is a totally optional switch, basically for tuning freaks like me.
- `-y [level]` This option activates Bzip2 archive compression, which by default is turned off. You can even specify a numeric compression level, which goes from 0 (no compression) to 9 (best compression, slow processing). Bzip2 by default uses 6, which is the best speed/compression ratio for most files. I don't specify compression level, 6 is fine for me.
- `-s 600M` Here comes DAR's slice feature. The specified size of 600 Megabytes is the maximum file size DAR will create. If your backup is bigger, you will end up with different backup files each with a slice number before the file extension, so you can save each file to a different unit of your backup media (floppies, zip, CDROM, etc). My backups are much smaller than this size, and I keep this switch just to be safe if I happen to create a big file in my home directory and forget to delete it. If this switch is useful for you, check DAR's manual for the `-S` switch too.
- `-D` Stores directories excluded by the `-P` option or absent from the command line path list as empty directories. This is helpful when you are recovering a backup from scratch, so you don't have to create manually all the excluded directories.
- `-R /` Specifies the root directory for saving or restoring files. By default this points to the current working directory. We are doing a *system backup* here, so it will be the root directory.

- `-c ‘date -I‘_data` This is the mandatory switch I talked of before, and it means to create a backup archive. For those who don't understand what follows, `‘date -I‘` is the shell's back tick expansion. In short, `date -I` will provide a date as YYYY-MM-DD format. With back ticks and used as a parameter, the output of the command will be used as a string of the parent command. This way you can create backup archives with the creation date embedded in the name. If you still don't understand what I'm talking about, try to run the following from the command line:

  ```
  echo "Today's date is ‘date -I‘"
  ```

- `-Z file_pattern` Using normal file name globing you can specify patterns of files you want to store in your archive without compression. This only has sense if you use the `-y` switch. Compressing compressed files only yields bigger files and wasted CPU time.
- `-P relative_path` With this switch you tell DAR which paths you don't want to store in your backup archive. Here you want to put the home directory (I'm the only user on this machine, there are a few more, but they are for testing/system purpose), system directories which aren't really physical files like `proc`, other drives you may have mounted under `mnt` (most notably the drive you are putting the backup file), etc, etc. Note that the paths you specify must be relative to the path specified by the `-R` switch.

That wasn't so hard. Check DAR's manual page for more useful switches you might want to use. And here's the command line I'll be running as a plain user inside my home directory:

```
dar -m 256 -y -s 600M -D -R /home/gradha -c ‘date -I‘_data \
   -Z "*.gz" -Z "*.bz2" -Z "*.zip" -Z "*.png" \
   -P instalacion_manual -P Mail/mail_pa_leer
```

Nothing new under the sun. As you see, most of the command line is identical to the other one, I only change the name of the directories I want to exclude with `-P` and the root directory with the `-R` switch.

Making differential backups with DAR

Once you have a full backup you can create a differential backup. The first differential backup has to be done using the full backup as reference. The following differential backups use the latest differential backup as reference. Here's the command line for a *system* differential backup:

```
dar -m 256 -y -s 600M -D -R / -c ‘date -I‘_diff -Z "*.gz" \
   -Z "*.bz2" -Z "*.zip" -Z "*.png" -P home/gradha -P tmp \
   -P mnt -P dev/pts -P proc -P floppy -P burner -P cdrom \
   -A previous_backup
```

- `-c ‘date -I‘_diff` I only change the name of the file, cosmetic purpose.
- `-A previous_backup` This new switch is used to tell DAR where is to be found the previous backup so it can create a differential backup instead of a full backup. The only thing you have to take care of is that you don't specify slice neither extension in the file name, otherwise DAR will make you an interactive question at the command line.

The user command line is exactly the same. Here it is for completeness:

```
dar -m 256 -y -s 600M -D -R /home/gradha -c `date -I`_diff \
   -Z "*.gz" -Z "*.bz2" -Z "*.zip" -Z "*.png" \
   -P instalacion_manual -P Mail/mail_pa_leer -A previous_backup
```

DAR has another nice feature we don't use here: *catalogues*. When you create a backup archive with DAR, internally it contains the data plus a *catalogue*. This *catalogue* contains information about what files were saved, their dates, their compressed size, etc. You can extract the *catalogue* and store it separately. Why would you want to do this? To set up networked differential backups.

In order to create a differential backup, you need to provide the previous backup so DAR can decide which files have changed or not. Doing this can be expensive in bandwidth if you work with a network. Instead, after you create a backup, you can extract the *catalogue* and send it to the machine doing the backups. Next time, you can use this file with the `-A` switch, and it will all work as if the complete file was there.

This can be also useful if you use slices, because the *catalogue* is created from the first and last slice. It's more comfortable to pass a single file to the backup command rather than having to carry the disks of your previous backup with you.

Setting up some scripts to automate the process

As said before, now it's the time to put our backup solution under cron. Place the following executable script for *system* backup under `/root/dar_backup.sh`:

```
#!/bin/sh

DIR=/var/backups/system
FILE=${DIR}/`/bin/date -I`_data
# Commands
/usr/local/bin/dar -m 256 -y -s 600M -D -R / -c $FILE -Z "*.gz" \
   -Z "*.bz2" -Z "*.zip" -Z "*.png" -P home/gradha -P tmp \
   -P mnt -P dev/pts -P proc -P floppy -P burner \
   -P cdrom -P var/backups > /dev/null
/usr/local/bin/dar -t $FILE > /dev/null
/usr/bin/find $DIR -type f -exec chown .gradha \{\} \;
/usr/bin/find $DIR -type f -exec chmod 440 \{\} \;
```

Some things to notice:

- DIR is the variable which holds the destination directory.
- FILE will hold the path to today's backup file.
- I use full paths for the commands because my root account doesn't have all of them included in the default environment. This is potentially a security risk. Ideally you would like to compile DAR as root and keep your binaries where you make them so nobody can touch them. And run Tripwire over them too.
- DAR generates statistics after each run. We don't want them in our cron because it will generate unnecessary mail. Only `stdout` is redirected to `/dev/null`. Errors will be reported and a mail generated if something goes wrong.

- The last two `find` commands are optional. I use them to change file ownership to a normal user, which will later create the backup. Again, another security risk. root should backup that from root, and users should backup their stuff. But with a mono user system, I don't care. If some intruder is good enough to go through my firewall and account passwords to take a look at my backups, I'm already screwed.

Now place the following nearly identical script for differential backups under `/root/dar_diff.sh`:

```
#!/bin/sh

DIR=/var/backups/system
FILE=${DIR}/‘/bin/date -I‘_diff
PREV=‘/bin/ls $DIR/*.dar|/usr/bin/tail -n 1‘
/usr/local/bin/dar -m 256 -y -s 600M -D -R / -c $FILE -Z "*.gz" \
   -Z "*.bz2" -Z "*.zip" -Z "*.png" -P home/gradha -P tmp -P mnt \
   -P dev/pts -P proc -P floppy -P burner -P cdrom \
   -P var/backups -A ${PREV%%.*} > /dev/null
/usr/local/bin/dar -t $FILE > /dev/null
/usr/bin/find $DIR -type f -exec chown .gradha \{\} \;
/usr/bin/find $DIR -type f -exec chmod 440 \{\} \;
```

The only two changes are the addition of the `-A` switch and the generation of the PREV variable with a complicated command line. Let's see what this command line does:

- First the `ls` command creates a list of the files with `.dar` extension in the backup directory. This output is piped to the next command.
- By default `ls` displays files alphabetically. `tail` is used to get the last file with the `-n 1` switch, which says to display only the last line.
- DAR wants to operate on filenames without slice number and extension. This means that if we don't get rid of the tail, DAR will stop the operation and ask an interactive question to the user, defeating the purpose of automation. We separate the complete filename with a Bash feature called parameter expansion. There are several possible expansions, you can type `man bash` to see all of them. The one using `%%` will remove the longest tailing pattern that matches whatever goes after the `%%`. The result is the base name we want to pass DAR.

We only have to put these two scripts under cron control. This is what we have to type after `crontab -e`:

```
15 0 2-31 * * ./dar_diff.sh
15 0 1    * * ./dar_backup.sh
```

Look up in `man -S 5 crontab` the syntax of the command. In short, those two lines tell cron to run the scripts 15 minutes past midnight. `dar_backup.sh` will be run only the first day of the month. The other script will be run all the other days.

Here are the backup scripts for your users. They are the same, changing only switches to the DAR command and paths:

```
#!/bin/sh
# dar_backup.sh

DIR=/var/backups/gradha
FILE=${DIR}/‘/bin/date -I‘_data
# Commands
/usr/local/bin/dar -m 256 -y -s 600M -D -R /home/gradha -c $FILE \
```

```
   -Z "*.gz" -Z "*.bz2" -Z "*.zip" -Z "*.png" \
   -P instalacion_manual -P Mail/mail_pa_leer > /dev/null
/usr/local/bin/dar -t $FILE > /dev/null
/usr/bin/find $DIR -type f -exec chmod 400 \{\} \;

#!/bin/sh
# dar_diff.sh

DIR=/var/backups/gradha
FILE=${DIR}/‘/bin/date -I‘_diff
PREV=‘/bin/ls $DIR/*.dar|/usr/bin/tail -n 1‘
/usr/local/bin/dar -m 256 -y -s 600M -D -R /home/gradha -c $FILE \
   -Z "*.gz" -Z "*.bz2" -Z "*.zip" -Z "*.zip" \
   -P instalacion_manual -P Mail/mail_pa_leer \
   -A ${PREV%%.*} > /dev/null
/usr/local/bin/dar -t $FILE > /dev/null
/usr/bin/find $DIR -type f -exec chmod 400 \{\} \;
```

Don’t forget to add the required crontab entries for your user pointing to the appropriate path.

Recovering your backup to a clean machine

When the time comes to restore your backup, depending on what you saved you will have a full backup of one month plus differential backups up to the last time you managed to make. The restoration process is very simple, it’s the same as described on the first chapter (Simple DAR usage), only you have to do it first for the full backup, and then for the differential ones. This can be boring, so here’s another shell script you can save with your backup files:

```
#!/bin/sh

if [ -n "$3" ]; then
   CMD="$1"
   INPUT="$2_data"
   FS_ROOT="$3"
   $CMD -x "$INPUT" -w -R "$FS_ROOT"
   for file in ${INPUT:0:8}*_diff*; do
      $CMD -x "${file:0:15}" -w -R "$FS_ROOT"
   done
   echo "All done."
else
   echo "Not enough parameters.

Usage: script dar_location base_full_backup directory

Where dar_location is a path to a working dar bi-
nary, base_full_backup
is a date in the format ’YYYY-MM-
DD’, and directory is the place where
you want to put the restored data, usually ’/’ when run as root."
```

```
fi
```

The script is pretty self explicative. The only things you would care is the `-w` switch, which tells DAR to overwrite found files. This is necessary for differential backups. Oh, and place the script in the same directory where you put your backup files. Here's an usage example:

```
./recover.sh /usr/local/bin/dar 2003-10-01 /tmp/temp_path/
```

Try to run that as a normal user with a few of your backup files. You can put the result in a temporary directory, so the nice thing is you don't have to wipe your hard disk to test it.

Adding checks to the backup scripts

Denis Corbin suggests that the scripts creating the backups could verify the exit status of the DAR command. For the purpose of these very simple scripts this is not critical because DAR itself will bail out with an error message, and cron will report any output through mail (something which doesn't happen if everything goes right).

However, testing the exit status can be useful if you are testing the scripts interactively and want to know which commands are executed:

```
#!/bin/sh

DIR=/var/backups/system
FILE=${DIR}/‘/bin/date -I‘_data
# Commands
if /usr/local/bin/dar -m 256 -y -s 600M -D -R / -c $FILE -
Z "*.gz" \
      -Z "*.bz2" -Z "*.zip" -Z "*.png" -P home/gradha -P tmp \
      -P mnt -P dev/pts -P proc -P floppy -P burner \
      -P cdrom -P var/backups > /dev/null ; then
   if /usr/local/bin/dar -t $FILE > /dev/null ; then
      echo "Archive created and successfully tested."
   else
      echo "Archive created but test FAILED."
   fi
else
   echo "Archive creating FAILED."
fi
/usr/bin/find $DIR -type f -exec chown .gradha \{\} \;
/usr/bin/find $DIR -type f -exec chmod 440 \{\} \;
```

You can test this version easily running the script and killing the DAR process from another terminal or console with `killall dar`. That will force the termination of the DAR process and you will see that one of the failure branches is reached in the backup script.

Another possible use of testing the status code could be to remove incomplete archives from the hard disk if something went wrong, trigger additional external commands when something fails, or avoid testing the created archive when you know that the first command already failed. The latter can be done easily concatenating both the creation and testing commands with `&&` in a single line. That will tell the shell to run both commands as a sequence and avoid running the second if the first failed.

However, if a power failure happens in the middle of a backup, this version of the script would still leave dangling invalid archives. To prevent this you could enhance the script to

do a *positive verification*. This means creating the backup in a temporary directory along with a `*.valid` file if the successful branch of the script is reached.

With this strategy, another cron script monitoring the directory where the temporary backups are placed would move to the final backup directory those archives which have a `*.valid` file, deleting all other whose last modification timestamp is older than one hour.

Ideas for the future

I'm not going to implement these soon, because I'm very lazy, but if you are one of those hyperactive hackers, here are some things which would be nice:

- Unify both the main and differential scripts into a single one, so if the script is run and there is no main backup for the current month, the main backup will be created. Useful if your machine happens to be down during the time the monthly backup is done.
- Upgrade the scripts to generate daily a CDROM image with cdrecord and burn it automatically to a rewritable disc placed in your machine. So if your whole hard disk is trashed, you still have the last backup on removable media. Of course, this is limited and cannot be automated if your backup spans more than one CDROM. Do the same for ZIP/JAZZ/whatever you have.
- Integration of generated backups with a mini Knoppix bootable distribution. Or any other floppy distribution which can be booted from CDROM. So you have a recovery CDROM with tools to format your hard disk, and near it you have a fresh backup to restore a working machine.
- Synchronisation of backup directories through Internet with remote hosts. Even if the whole machine is burnt physically along with your house, you have up to date backups somewhere else. Could be done easily with programs like rsync through ssh running in a cron job.
- Factor common parameters into a separate file and include it from your scripts using DAR's -B switch. For instance:

  ```
  $ cat > /var/backups/system/common.dcf
  -m 256 -y -s 600M -D -R / -Z "*.gz" -Z "*.bz2" -Z "*.zip" \
  -Z "*.png" -P home/gradha -P tmp -P mnt -P dev/pts \
  -P proc -P floppy -P burner -P cdrom -P var/backups
  ```

  Later on you could use this in the script:

  ```
  DIR=/var/backups/system
  FILE=${DIR}/‘/bin/date -I‘_data
  # Commands
  /usr/local/bin/dar -B ${DIR}/common.dcf -c $FILE > /dev/null
  /usr/local/bin/dar -t $FILE > /dev/null
  /usr/bin/find $DIR -type f -exec chown .gradha \{\} \;
  ```

  Which you can reuse in the differential version too!

In fact, clever people out there have already started making such scripts for themselves and are not afraid to share them. To avoid cluttering this mini-howto I'm going to store them *as-is* at my web page: http://gradha.sdf-eu.org/dar_scripts/.

Feel free to send me your own improvement and I'll add it to the directory. Whether you are sending a single script file or `.tar.gz` with a whole backup suite, please add a simple `.txt` file which I'll put near the file, so people can read what the file does before downloading.

Please use English in your description, and don't forget to put your name and email so people can send you bugfixes or improvements!

The end

And that's the whole *magic*. If you have problems, something is unclear or wrong (which is worse), drop me an email. If you find this document useful and want to translate it, send me a translation of the file `source.en.txt` so I can distribute it along this version and users can find easily their localized version. Talking about locations, you should be able to get the source of this document from my personal home page (link at the beginning of the document).

Enjoy!

Generated by Docutils from reStructuredText source.