Some interesting tools

use them or don't use them

LIA
Universidade de Vigo
Escola Superior de Enxeñaría Informática
E–32004 Ourense

http://lia.ei.uvigo.es
mailto:formella@uvigo.es

Contact: Arno Formella

Reference:	LIA-DOC-TOOLS
Version:	1.2
Date:	15/03/2013
Pages:	17

Contents

1 Introduction

This document describes in a ready-to-use manner very convenient tools for the daily work. All scripts and configurations files can be downloaded from our web-site `http://lia.ei.uvigo.es`. Comments are welcome.

1.1 Purpose and scope

Although the descriptions in this document are tailored for a GNU/Linux environment, almost all tools are available for Windows and Mac systems as well.

Whenever you install a new working environment you should consider making these tools available to the user. The installation process is not included here, each tool describes the installation process in its own documentation.

The text is neither a tutorial nor a handbook; there are other sources for this type of documents. Rather it describes a complete minimum setup of the tools which in most typical environments is sufficient to start with. It is clearly highlighted which parts must be adapted to fit personal needs.

Especially when you are working with or within the **LIA** research group, you should know and use these tools.

1.2 Revision history

Version 1.2:
- Sections for **dar**(backup) recovering added.
- Section for **make**(task automation) added.

Version 1.1:
- Section for formatting C–source files added.
- Error in `cd`–command in setting up an empty repository in the **git** section fixed.

Version 1.0:
This is the initial document written in summer of 2010.

1.3 References

2 dar: Backup

Hardware fails, that is a fact. Therefore, you must backup at least all files that cannot be restored by other means, i.e., escentially you must backup your own work. This section briefly describes how to use the disk archiving tool **dar** to make first a full backup and then differential backups of your home directory. As principal backup media, we assume an external USB harddrive.

First of all, doing a good backup is not just copying files, rather you want to be able to restore a true snapshot of a file system including correct ownership for the files, correct time stamps, soft- and hardlinks, and much more. Hence, you need a tool, for instance **dar**.

dar, available at `http://dar.linux.free.fr` or `http://dar.sourceforge.net` is a GPL (v2) licensed software package to make backups of entire directory trees. **dar** is a shell command tool. There are GUIs available. The documentation is extensive and well done.

dar can be tailored to make effective backups of entire systems. The main author is Denis Corbin, this document deals with version 2.3.10.

In order to make your backups, in the on-going you will see the most basic steps; feel free—with the help of the documentation of **dar**—to extend the suggestions.

● Note that we either store a full backup or a differential backup towards such a full backup, i.e., we do not iterate differential backups (a choice you can take if you like, **dar** is prepared for doing that).

2.1 Setup of the configuration file

dar can use a configuration file named `.darrc` in the users home directory (note, there are more options available to do the job). The listing below shows such a configuration file where we do the following:

- We divide the backup into slices of at most 600MB (`-s` option), so we can write them on CD whenever we need.
- We compress all files for which compression pays–off (`-z` and `-Z` options), so we save some disk space.
- We exclude temporary files and some files for security reasons (`-X` and `-P` options), so we save disk space and don't exhibit secrets.
- We make use of case-insensitive filenames and regular expressions. (`-an` and `-ar` options), so we are a bit more flexible.
- We maintain excluded directories as empty directories (`-D` option), so at least we know there has been something.
- We restore without any option (default run), so at least this will be easy after a crash.

Lines starting with a `#` are comment lines. The entries are sorted within groups in alphabetic order.

```
#
# (c) copyright 2010 Arno Formella
# formella@ei.uvigo.es,  lia.ei.uvigo.es
#

# we run dar normally
#    to generate a full backup
#       dar -c ${1?}_`date -I`_full -R ${HOME?}
#    to generate a  differential backup
#       dar -c ${1?}_`date -I`_diff -R ${HOME?} -A ${2?}
#    to generate a catalog
#       dar -C ${1?}_`date -I`_ctlg -A ${2?}

# options for all command variants
all:
# slice always so we can grab on CD-ROM if we like
  -s 600M

# -c option, creation of an archive
create:
# compress with gzip
  -z
# but exclude from compression the following file types
# (they are usually compressed well enough)
# take the extension case-insensitive
  -an
  -Z "*.asf"
  -Z "*.avi"
  -Z "*.bz"
  -Z "*.bzip"
  -Z "*.bzip2"
  -Z "*.bz2"
  -Z "*.deb"
  -Z "*.divx"
  -Z "*.fdf"
  -Z "*.gif"
  -Z "*.gz"
  -Z "*.gzip"
  -Z "*.jar"
  -Z "*.jpg"
  -Z "*.mov"
  -Z "*.mp3"
  -Z "*.mp4"
  -Z "*.mpeg"
  -Z "*.mpg"
  -Z "*.ogg"
  -Z "*.pdf"
  -Z "*.rar"
  -Z "*.rpm"
  -Z "*.swf"
  -Z "*.sxw"
  -Z "*.tbz"
  -Z "*.tgz"
  -Z "*.tif"
  -Z "*.tiff"
  -Z "*.wma"
  -Z "*.wmv"
  -Z "*.zip"
  -Z "*.7z"
  -Z "*.Z"
  -acase
# create empty directories for excluded directories
  -D
# exclude the following configuration directories of certain
# applications, caches etc.
# because we don't care, after a crash we would
# install the applications anyway from scratch
  -P ".adobe/"
  -P ".agdserver/"
  -P ".fontconfig/"
  -P ".googleearth/"
  -P ".kde/"
  -P ".local/share/Trash/"
  -P ".loki/"
  -P ".macromedia/"
  -P ".mcop/"
  -P ".openoffice.org2/"
  -P ".qt/"
  -P ".strigi/"
  -P ".thumbnails/"
# exclude some things for security reasons
  -P ".ssh/"
  -X ".bash_history"
  -X ".forward"
  -X ".ICEauthority"
  -X ".lesshst"
  -X ".netrc"
  -X ".profile"
  -X ".recently-used"
  -X ".recently-used.xbel"
  -X ".sudo_as_admin_successful"
  -X ".viminfo"
  -X ".Xauthority"
  -X ".xsession_errors"
# exclude temporary files and automatically generated
# documentation files as well as all temporary files of
# compilers and the tex system
  -X "diff.txt"
  -X "err.txt"
  -X "ls.txt"
  -X "*~"
  -X "*.bak"
  -X "*.swp"
  -X ".gnuplot_history"
# here we switch to regular expression parsing for the options
  -ar
  -P ".*/\._d/"
  -P ".*/Cache/"
  -P ".*/\.libs/"
  -P ".*/doc/html/"
  -P ".*/doc/latex/"
# we switch back to normal file globbing
  -ag
  -X "*.a"
  -X "*.aux"
  -X "*.blg"
  -X "*.d"
  -X "*.dar"
  -X "*.dvi"
  -X "*.fls"
  -X "*.glg"
  -X "*.glo"
  -X "*.gls"
  -X "*.haux"
  -X "*.htoc"
  -X "*.idx"
  -X "*.ilg"
  -X "*.ind"
  -X "*.ist"
  -X "*.lo"
  -X "*.lod"
  -X "*.lof"
  -X "*.log"
  -X "*.lot"
  -X "*.maf"
  -X "*.mlf*"
  -X "*.mlt*"
  -X "*.mtc*"
  -X "*.nav"
  -X "*.o"
  -X "*.out"
  -X "*.plf*"
  -X "*.ptc*"
  -X "*.slf*"
  -X "*.slt*"
  -X "*.snm"
  -X "*.so"
  -X "*.stc*"
  -X "*.toc"
  -X "*.vrb"
# SRTM terrain files a huge and usually backuped somewhere else
  -X "*.hgt"
  -X "*.hgt.zip"
# exclude dar archives and iso-CD-burn files, too
  -X "*.*.dar"
  -X "*.iso"

# -x option, extraction of the files from an archive
# no options, it should work as simple as possible
extract:
```

● Feel free to modify the file patterns so they match your specific needs.

2.2 Invocation of **dar**

2.2.1 Full backup

Assume you want to backup your home directory on machine `machine` to an external USB drive mounted at `/media/disk` in the already existing directory `backups/machine`. To make a full backup and a lookup catalog, we run the command

```
source dar_bck_full.sh /media/disk/backups/machine/machine
```

with the following script `dar_bck_full.sh`

```
#
# (c) copyright 2010 Arno Formella
# formella@ei.uvigo.es, lia.ei.uvigo.es
#

# generate full backup
dar -c ${1?}_`date -I`_full -R ${HOME?}
# generate catalog
dar -C ${1?}_`date -I`_ctlg -A ${1?}_`date -I`_full
```

So we generate all slices of the full backup archive in the directory `/media/disk/backups/machine`.

The slices are named, e.g.,
`machine_2008-08-28_full.1.dar`, `machine_2008-08-28_full.2.dar`, etc.,
and the catalog for faster future differential backups is named, e.g.,
`machine_2008-08-28_ctlg.1.dar`.

2.2.2 Differential backup

To make a differential backup, we run the command

```
source dar_bck_diff.sh \
  /media/disk/backups/machine/machine \
  /media/disk/backups/machine/machine_YYYY-MM-DD_ctlg
```

where you have to substitute the sequence `YYYY-MM-DD` according to the corresponding date of the base full backup, with the following script `dar_bck_diff.sh`

```
#
# (c) copyright 2010 Arno Formella
# formella@ei.uvigo.es, lia.ei.uvigo.es
#

# generate differential backup
dar -c ${1?}_`date -I`_diff -R ${HOME?} -A ${2?}
```

2.2.3 Restoring the backup

To restore a backup back onto your system just use twice the script `dar_bck_rest.sh`

```
#
# (c) copyright 2010 Arno Formella
# formella@ei.uvigo.es, lia.ei.uvigo.es
#

# restore backup to home directory
dar -x -w -r ${1?} -R ${HOME?}
```

where the first time you specify as argument the base name of the full backup and the second time the base name of the differential backup. The additional options `-w` means to overwrite files but `-r` confines the writing to overwriting older files by newer ones, but not the other way round.

2.2.4 Restoring some files only

To restore or recover a specific file from your backup (for instance, in case it was deleted/lost for some reason) run the script `dar_bck_reco.sh`

```
#
# (c) copyright 2010 Arno Formella
# formella@ei.uvigo.es, lia.ei.uvigo.es
#
```

```
# restore backup to home directory
dar -R ${HOME} -x ${1?} -v -g ${2?}
```

with the first argument specifing the differential backup and the second argument specifying the file to be recovered with its relative path name starting from your home directory. If you see a line like

```
restoring file: /home/denis/my_precious_file
```

you are done, otherwise run the script again but now with the full backup as first argument.

For more sophisticated recovering actions, please see the manual of **dar**and consider using `dar_manager`.

3 **git**: Version control

Version control (or revision control) and backup are in certain aspects quite similar, however, their main objectives are different: the main purpose of a backup is to be able to recover smoothly after a more or less severe crash; version control means that you can go back to certain points in time and restore the ancient snapshot of a development, possibly maintaining different branches within the same project.

There are many version control software tools available. **git** is one of them with one outstanding characteristic: the possibility of serverless distributed colaboration.

git, available at `http://www.git-scm.com` is a GPL (v2) licensed software package for fast, scalable, distributed revision control. **git** is a shell command tool. There are GUIs available. The documentation is extensive and well done. There are migration tools to/from other version control systems available. The main author is Linus Torvalds, this document deals with version 1.7.2.1, maintainer Junio C. Hamano.

In order to control your devolopment, in the on-going you will see the most basic setups and simple scripts; feel free—with the help of the documentation of **git**—to extend the suggestions.

3.1 Initialization of **git**

The initialization of **git** is—at least for a minimum configuration—quite simple. There are some globally available settings to be done (you can even skip this part, then **git** will guess the settings from your system and user installation). The most basic are telling **git** your user name and your email address, so the repository will have this information ready for other users possibly working with it.

```
git config user.name <your name>
git config user.email <your email>
```

The **git**–command stores such configuration data in a user local configuration file named `.gitconfig` being placed in your home directory. See the **git** documentation for more details about the possible content of this file.

To proceed change the directory to your working directory and initialize a new empty **git** repository:

```
cd <workdir>
git init
```

This directory can be an empty directory when you just start a new project, or it can be an already populated one that you want to put under **git** version control. The command generates a subdirectory named `.git` and places some intial configuration data in there. Later the directory will contain the entire repository for the project. Mostly, you don't have to deal directly with the content of this directory, however, make sure that it is part of your backup.

Edit in the working directory a `.gitignore`–file which will contain all file patterns to be ignored by **git**. An example of a `.gitignore`–file for a directory used for C++ programming is:

```
#
# (c) copyright 2010 Arno Formella
# formella@ei.uvigo.es, lia.ei.uvigo.es
#

# Note: take care of trailing white space characters at end
# of pattern git does not ignore them and you may get
# unexpected effects.

# the compiler generated files
*.a
*.d
*.o
*.lo
*.so
*.Tpo

# temporary files or links for compilation
.depend
.make.state
CONFIG

# the entire documentation files, but the logos
*/src/doc/latex/
*/src/doc/html/*
*/src/doc/html/*/
!*/src/doc/html/doxy_*.png

# backup files of different tools
*~
*.bak
*.swp

# results of diff and error output, I usually use
diff.txt
err.txt

# archive files
*.deb
*.tar
*.tar.bz*
*.tar.gz
```

```
*.tbz
*.tgz

# autoconf configuration files
config.mak
autom4te.cache
config.cache
config.log
config.status
config.mak.autogen
config.mak.append
configure
```

As you see, lines starting with the `#`–symbol serve as comment lines. You can use wildcards as usual. An example of a `.gitignore`–file for a LaTeX text working directory is:

```
#
# (c) copyright 2010 Arno Formella
# formella@ei.uvigo.es, lia.ei.uvigo.es
#

# Note: take care of trailing white space characters at end
# of pattern git does not ignore them and you may get
# unexpected effects.

# Ignore all files latex tools generate as output.
# Usually these files can be re-generated
# with the exception of implicite dates in the output.
# So take care that you use absolute dates if a future
# re-run of latex should produce exactly the same output
*.bbl
*.dvi
*.gls
*.html
*.ind
*.pdf
*.ps

# Do not ignore scanned, signed, or marked final pdf files.
!*_final.pdf
!*_scanned.pdf
!*_signed.pdf

# Ignore all temporary files generated by a latex system.
*.aux
*.blg
*.fls
*.glg
*.glo
*.haux
*.htoc
*.idx
*.ilg
*.ist
*.lod
```

```
*.lof
*.log
*.lot
*.maf
*.mlf*
*.mlt*
*.mtc*
*.nav
*.out
*.plf*
*.plt*
*.ptc*
*.slf*
*.slt*
*.snm
*.stc*
*.toc
*.vrb

# Ignore tree of make tool
._d/
```

● Note that you must not use trailing white spaces in the patterns, because **git** will interpret these characters as part of the pattern which may not be what you want.

● Note that the excluded files and directories should be related to your usual way of working and should be adapted to your needs and the tools you employ.

3.2 Working with **git**

This section briefly describes how to put files under **git** control and how to commit modifications of the files such that they are recorded in the repository.

To put all files that do not match any pattern in your `.gitignore` file under control of **git** execute:

```
git add .
git commit -a
```

The `add`–command adds all files to the index (marking files to be handled) and the `commit`–command store the necessary information into the repository. Instead of specifying all changed and non-excluded files, you can work with individual files as well:

```
git add <filename>
git commit <filename>
```

3.3 Using **git** to move your files from system to system

We use **git** to maintain a repository on a USB-memory pen drive which serves to transfer files from one computer to another.

3.3.1 Preparing a USB memory pen drive

You might want to install an `ext3` file system on the pen drive and give it a name. In most cases, such devices are delivered with a `FAT` file system. You can skip this formating of the pen drive, however, a GNU/Linux file system has certain advantages over a `FAT` system.

● Be aware that you will loose all information stored on the device, so take precautions beforehand.

Plug in your pen drive in a free slot. Once the device is mounted run

```
df
```

In the listing that appears the first column indicates the device and the last column the directory name where the device is mounted. Find your pen drive in the listing (a good candidate is `/media/disk` which is used below in the examples).

Unmount the device:

```
sudo umount /media/disk
```

Format the device, give it a name, and synchronize the system:

```
sudo mkfs.ext3 /dev/sdb1
sudo e2label /dev/sdb1 liadisk
sync
```

● **Warning:** be careful to use the correct device, otherwise you may destroy relevant data on other devices.

Finally you can re-mount the device, or just remove and re-plug-in.

3.3.2 Setting up an empty repository

● First setup a bare and shared repository on the pen drive. As example we use here the repository name `lia.git`. Note that **`git`**-repositories by convention use the `.git`-extension. We assume that an `ext3` file system with root previleges has been installed (see previous section), hence, we create the directory of the repository as root and change its mode so all users can act on it.

```
sudo mkdir /media/liadisk/lia.git
sudo chmod -R 0777 /media/liadisk/lia.git
cd /media/liadisk/lia.git
git --bare init --shared
```

3.3.3 Cloning the empty repository onto your computers

Clone the empty **`git`** repository that we created on the pen drive onto one of your computers (and ignore the warning emitted by **`git`**):

```
git clone /media/liadisk/lia.git
```

This will create at the point in the file system where you run the command a directory named `lia` containing just the `.git` subtree, but still no more files.

Either start a new project there or populate the directory with files your want to put under **git** control. Don't forget to include an appropriate `.gitignore`–file. Work as usual with **git** adding and committing files. Once everything is finished, push the changes to the pen drive, for instance with:

```
<work-with-files>
git add .
git commit -a
git push origin master
```

This will update the repository on the pen drive with your local repository content, i.e., you will have an exact copy.

```
git clone /media/liadisk/lia.git
<work-with-files>
git add .
git commit -a
git push origin master
```

This again will update the repository in the pen drive with your local repository on the second machine.

From now on, you work always in the following way:

- Plug-in your pen drive.
- Pull the repository from the pen on your computer. If nothing has been modified, you get the appropriate message. If you forgot to plug-in your pen drive, you will get the appropriate error message. If you must merge, you are prompted to do so.
- Work with your local repository as usual with **git**.
- Push the changes to the pen drive.

The whole process with **git**–commands (as example, there is more...):

```
git pull
<work-with-files>
git add .
git commit -a
git push origin master
```

Note that we created three identical repositories. We can either work as described only on one computer at a time using the pen drive to synchronize whenever we switch machine, or we can develop on the different machines, possibly even with different users, different branches of the project which—at the points whereever we wish—can be merged (see **git** documentation for more details of these more complex operations).

4 **make**: task automation

4.1 Makefile for LaTeX documents

4.2 Makefile for C++ compilation

5 **indent**: formatting of C–source files

The classical **indent**command line program can be used to format C–source files (`.c` and .h files) more or less similar to the programming style of **LIA** for C++. You use the following indent profile (i.e., `.indent.pro` in your home directory).

```
//
// (c) copyright 2010 Arno Formella
// formella@ei.uvigo.es,  lia.ei.uvigo.es
//

//     --blank-lines-after-commas                //      -bc
    --blank-lines-after-declarations          //     -bad
    --blank-lines-after-procedures            //     -bap
    --blank-lines-before-block-comments       //     -bbb
//     --braces-after-if-line                    //      -bl
    --braces-after-func-def-line              //     -blf
//     --brace-indent                            //      -bli
//     --braces-after-struct-decl-line           //      -bls
    --braces-on-if-line                       //     -br
    --braces-on-func-def-line                 //     -brf
    --braces-on-struct-decl-line              //     -brs
    --break-after-boolean-operator            //     -nbbo
//     --break-before-boolean-operator           //      -bbo
    --break-function-decl-args                //     -bfda
    --break-function-decl-args-end            //     -bfde
    --case-indentation2                        //     -clin
    --case-brace-indentation0                  //     -cbin
    --comment-delimiters-on-blank-lines       //     -cdb
//     --comment-indentation                     //      -cn
    --continuation-indentation2                //     -cin
//     --continue-at-parentheses                 //      -lp
    --cuddle-do-while                         //     -cdw
//     --cuddle-else                             //      -ce
//     --declaration-comment-column              //      -cdn
    --declaration-indentation2                 //     -din
//     --dont-break-function-decl-args           //      -nbfda
//     --dont-break-function-decl-args-end       //      -nbfde
    --dont-break-procedure-type               //     -npsl
//     --dont-cuddle-do-while                    //      -ncdw
    --dont-cuddle-else                        //     -nce
//     --dont-format-comments                    //      -nfca
//     --dont-format-first-column-comments       //      -nfc1
    --dont-line-up-parentheses                //     -nlp
    --dont-left-justify-declarations          //     -ndj
    --dont-space-special-semicolon            //     -nss
```

```
        --dont-star-comments                    //      -nsc
         --else-endif-column1                    //      -cpn
        --format-all-comments                   //      -fca
        --format-first-column-comments          //      -fc1
//        --gnu-style                             //      -gnu
        --honour-newlines                       //      -hnl
//        --ignore-newlines                       //      -nhnl
        --ignore-profile                        //      -npro
         --indent-label1                         //      -iln
         --indent-level2                         //      -in
//        --k-and-r-style                         //      -kr
//        --leave-optional-blank-lines            //      -nsob
//        --leave-preprocessor-space              //      -lps
//        --left-justify-declarations             //      -dj
//        --line-comments-indentation             //      -dn
        --line-length80                           //      -ln
//        --linux-style                           //      -linux
        --no-blank-lines-after-commas           //      -nbc
//        --no-blank-lines-after-declarations     //      -nbad
//        --no-blank-lines-after-procedures       //      -nbap
//        --no-blank-lines-before-block-comments  //      -nbbb
//        --no-comment-delimiters-on-blank-lines  //      -ncdb
        --no-space-after-casts                  //      -ncs
//        --no-parameter-indentation              //      -nip
        --no-space-after-for                    //      -nsaf
        --no-space-after-function-call-names    //      -npcs
        --no-space-after-if                     //      -nsai
        --no-space-after-parentheses            //      -nprs
        --no-space-after-while                  //      -nsaw
        --no-tabs                               //      -nut
//        --no-verbosity                          //      -nv
//        --original                              //      -orig
         --parameter-indentation2                //      -ipn
         --paren-indentation0                    //      -pin
        --preserve-mtime                        //      -pmt
//        --preprocessor-indentation              //      -ppin
//        --procnames-start-lines                 //      -psl
//        --space-after-cast                      //      -cs
//        --space-after-for                       //      -saf
//        --space-after-if                        //      -sai
//        --space-after-parentheses               //      -prs
//        --space-after-procedure-calls           //      -pcs
//        --space-after-while                     //      -saw
//        --space-special-semicolon               //      -ss
//        --standard-output                       //      -st
//        --start-left-side-of-comments           //      -sc
//        --struct-brace-indentation              //      -sbin
        --swallow-optional-blank-lines          //      -sob
         --tab-size2                             //      -tsn
//        --use-tabs                              //      -ut
        --verbose                               //      -v
```

- Note that you must specify additionally all type definitions in your sources with the `-T`–option.
- See the **indent**–manpage for further details on how to invoke the formatter.

The generated files still have some *problems* in not fulfilling the rigorously the **LIA** style recommendations:

- The closing brace of function declarations and definitions, i.e., after the parameter declaration, is not put into a new line and not aligned to the suggested column.
- There are introduced to much empty lines between declarations.
- The variables are not declared all in their own line.
- The space between `switch` and the following parenthesis is not removed.
- Some comments after code is broken incorrectly when they spread over more than one line which may lead to uncompilable code.
- The pointer `*` is not placed directly after the typename.
- Some strings are not aligned nicely when used as literal parameters. Especially, they are not broken up into nicely readable parts.
- Code embedded into commants is formated as comment, not as code.
- Mixed C–style and C++–style comments are not always dealt with correctly.