Automating your backups on Unix-like systems (Linux and MacOS)

ALEX RAZOUMOV
alex.razoumov@westgrid.ca

Zoom controls

- Please mute your microphone and camera unless you have a question
- To ask questions at any time, type in Chat, or Unmute to ask via audio
- Raise your hand in Participants

- Email training@westgrid.ca

Disclaimer

- Many tools and possible workflows
- Everything presented today is based on my own workflow refined over the years
- Meant for backing up your own computer, or a cloud VM, not your data on Compute Canada HPC clusters
  - `/home`, `/project` are backed up already
- Double-check everything: these are your files, and you – only you! – are responsible
- View it as a template that you can customize, not the final script
- Tested on Linux and MacOS; these tools *will probably* work in bash shell in Windows (WSL, Cygwin, etc)
- You might have other preferences:
  - e.g. doing cron backups
  - I like to trigger backups manually as I can monitor them live for anomalies, and keep my backup drives disconnected most of the time

My take on the 3-2-1 backup strategy

- ✔ at least 3 data copies (one production + two backups)
- ✔ at least 2 different backup tools (really an array of tools packed into one function)
- ✔ at least 1 off-site copy (cloud or rotating drives off-site)

Tools

- Built-in tools, e.g. Time Machine on MacOS
  - ✔ easy to use
  - ✘ quite slow at times, sometimes takes hours for no obvious reason (updating its index)
  - ✘ results in millions of files that are hard to move around, weird permissions
  - ✘ requires a Mac to restore
- Commercial online tools, e.g. Backblaze
- Open-source tools: DAR , BORG , Restic, etc.

DAR

Not really a backup tool!
... was created as replacement for TAR

TAR limitations

- TAR is the most widely used archiving format on UNIX-like systems
  - first release in 1979
- Each TAR archive is a sequence of files
  - each file inside contains: a header + some padding to reach a multiple of 512 bytes + file content
  - EOF padding (some zeroed blocks) at the end of the TAR file
- Designed for sequential write/read on tape drives $\Rightarrow$ there is no index for random access to TAR contents $\Rightarrow$ extracting could be very inefficient
- Third-party tools can add indexing to TAR :
  - `https://github.com/devsnd/tarindexer` is a Python tool for indexing TAR files for fast access (writes a separate file)
  - RAT (`https://github.com/mcuadros/go-rat`) is an extension to embed the index at the end of the TAR file itself; any RAT-produced TAR file is compatible with standard TAR

Disk ARchiver (DAR)

http://dar.linux.free.fr

- Written from the ground up as a modern replacement to TAR
- Open-source, first release in 2002, actively maintained
  - full / incremental backup
  - each archive includes an index ⇒ fast file search / restore
  - build-in compression on a file-by-file basis
    - more resilient against data corruption
    - can avoid compressing already compressed files, e.g. `-Z "*.mp4" -Z "*.gz"`
  - strong encryption
  - can split archives at 1-byte resolution
  - supports extended file attributes, sparse files, hard and symbolic/soft links
  - can detect corruption in both headers and saved data, recover with minimal data loss
  - can merge two archives into a new one, e.g. can convert full+incremental backups to full
- Full DAR - TAR comparison http://dar.linux.free.fr/doc/FAQ.html#tar

Installing DAR

- Compile your own

```
brew install libgcrypt
wget https://downloads.sourceforge.net/project/dar/dar/2.6.13/dar-2.6.13.tar.gz
unpack and cd there
make clean distclean 2> /dev/null
export LDFLAGS="-L/usr/local/lib␣$LDFLAGS"
export CPPFLAGS="-I/usr/local/include␣$CPPFLAGS"
./configure --prefix=/path/to/installation --enable-libgcrypt --enable-mode=64
make
make install-strip
```

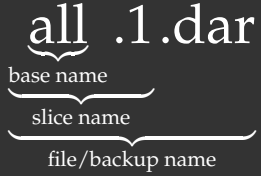
- Use your package manager
- DAR on Compute Canada systems

```
[user@system:~]$ module load StdEnv/2020
[user@system:~]$ which dar
/cvmfs/soft.computecanada.ca/gentoo/2020/usr/bin/dar
[user@system:~]$ dar --version
dar version 2.5.11, Copyright (C) 2002-2052 Denis Corbin
...
```

Seeking time

Watch our May 2019 webinar for DAR - TAR speed comparison on multi-GB files
`https://westgrid.github.io/trainingMaterials/tools/rdm`

Manual archiving and extracting



- Let's quickly generate 134MB of random data

```
cd ~/tmp
mkdir workspace && cd workspace
for num in $(seq -w 000 999); do
  echo $num
  # generate a binary file (8-264)kB in size
  dd if=/dev/urandom of=test"$num" bs=8 count=$(( RANDOM + 1024 ))
done
```

- Create a basic DAR archive

```
cd ~/tmp
dar -w -c all -g workspace              # create DAR archive all.1.dar
dar -l all                              # list its contents
mkdir restore
dar -R restore/ -O -w -x all -v -g workspace/test596   # extract one file
dar -R restore/ -O -w -x all -v -g workspace           # extract entire directory
```

  - `-w` will not warn before overwriting, `-O` will ignore ownership field when restoring, `-x` will extract

Incremental backups

- Create full backup `monday.1.dar`

```
/bin/rm all.1.dar
dar -w -c monday -g workspace
```

- Create first incremental backup `tuesday.1.dar`

```
dd if=/dev/urandom of=workspace/tue1 bs=8 count=$(( RANDOM + 1024 ))    # add a random fil
dar -w -A monday -c tuesday -g workspace
```

  - `-A` will use argument as a reference archive

- Create second incremental backup `wednesday.1.dar`

```
dd if=/dev/urandom of=workspace/wed1 bs=8 count=$(( RANDOM + 1024 ))    # add another file
dd if=/dev/urandom of=workspace/wed2 bs=8 count=$(( RANDOM + 1024 ))    # add another file
/bin/rm workspace/test999
dar -w -A tuesday -c wednesday -g workspace
```

Incremental backups (cont.)

- Check each backup for `workspace/test999`

```
dar -l monday | grep test999        # it is there
dar -l monday -g workspace/test999      # or can use the full path
dar -l tuesday -g workspace/test999      # it is there
dar -l wednesday -g workspace/test999     # shows REMOVED ENTRY
```

- Restore latest backup

```
dar -R restore -O -x wednesday   # restores only last incremental backup (wed1, wed2)
```

- To restore everything ending with the latest backup, start with the full (first) backup and go sequentially through all incremental backups

```
dar -R restore -O -w -x monday              # restore the full backup
dar -R restore -O -w -x tuesday
dar -R restore -O -w -x wednesday
```

Limiting the size of each slice

... to 10MB

```
dar -s 10M -w -c monday -g workspace      # from 134MB we get all.{1..14}.dar slices
dar -O -x monday                        # will extract all slices with `monday` basename
```

Backup

- `pack()` function is included into `functions202102.sh` ☞ let's study it

```
source /path/to/functions202102.sh
pack
pack show
pack 0    # create full backup
dd if=/dev/urandom of=test/item001 bs=8 count=$(( RANDOM + 1024 ))    # add random file
pack 1    # create first incremental backup
dd if=/dev/urandom of=test/item001 bs=8 count=$(( RANDOM + 1024 ))    # add random file
pack 2    # create second incremental backup
...
```

- You can always go back (likely not `'pack 0'`)

```
pack 1     # this will overwrite all1.1.dar and remove all{2..}.1.dar
pack 2
...
pack show
```

Backup (cont.)

- Each incremental backup has full index

```
dar -l backups/all2   # will list all files
```

  ⇒ no need for the full backup in $BDEST !!!

  ⇒ you can backup a large HDD/SSD to a much smaller USB drive!

- We'll add BORG in the next section

Restore

- `restore()` function is included into `functions202102.sh` ☞ let's study it

```
restore
restore -l test999          # pay attention to [Saved] tag
restore -n 2 workspace/test999   # will not restore as this file is not saved in that bac
restore -n 0 workspace/test999   # will restore but this is not safe

restore -x workspace/test999       # the safest option
restore -x workspace               # restore the entire directory
```

- DAR does not understand wildmasks
  ⇒ need to specify relative (to $BREF in `pack()`) directory or file path

Encryption

- Uncomment the encryption flag in `pack()` and source it

```
source /path/to/functions202102.sh
/bin/rm -rf backups/all*
pack 0    # provide password (same password twice)
pack 1    # provide password (password for 0, then password for 1 twice)
...

restore -x workspace     # will ask for a password
```

- **Note**: each backup has its own password, which may not be convenient ...

BORG

https://borgbackup.readthedocs.io

- Deduplicating backup program
- Open-source, first public release in July 2013, actively maintained
  - deduplication: each file split into chunks; only chunks that have never been seen before (in any file in any of the current / previous backups) are added to the repository
  - ⇒ moving / renaming files and directories will *not* result in extra copy
  - fast
  - 256-bit AES encryption
  - optional compression (choice of algorithms)
  - remote data storage over SSH: client only and client-server

Installing BORG

- Use `pip` with Python 3

```
pip install borgbackup      # e.g. into a virtual environment
```

Setting up your backup repository

```
rm ~/tmp/backups/*
```

- Initialize the repository

```
borg init --encryption=keyfile ~/tmp/backups          # enter a non-empty passphrase
```

  - this creates a private key in `~/.config/borg/keys`
  - you will need both your key and the passphrase to access the repository!

- Good idea to save the key in a safe place

```
borg key export ~/tmp/backups /some/safe/location     # can export to multiple locations
borg key export --qr-html ~/tmp/backups keyfile.html  # print QR code on paper
```

Manual backup and restore

- Backup your data

```
borg create --stats ~/tmp/backups::monday ~/tmp/workspace    # first backup
borg create --stats ~/tmp/backups::tuesday ~/tmp/workspace   # second backup
```

- Inspect your backup

```
borg list ~/tmp/backups             # list all backups in the repository
borg list ~/tmp/backups::tuesday    # list the contents of this specific backup
borg list --help
```

  - there is also '`borg info ...`' to get more detailed information and stats

- Restore your data

```
cd ~/tmp/restore
borg extract ~/tmp/backups::tuesday
```

  - this will restore the full backup (monday+tuesday) into the current directory
  - i.e. restores all files from the repository as of 'tuesday' (include all files from 'monday' except those that were deleted before 'tuesday')

- Restore a specific file or folder

```
borg extract ~/tmp/backups::tuesday Users/razoumov/tmp/workspace/test980
```

  - wildmasks do not seem to work

Delete and prune

- You can remove specific backups

```
borg delete ~/tmp/backups::monday    # remove only files that were deleted before 'tuesda
borg delete ~/tmp/backups::tuesday   # oops ... and now all backups are gone
```

- `borg prune` will delete all backups from a repository, except the ones that you specify, e.g.

```
borg prune -v --list $BDEST --keep-last=1    # keep only the last backup
borg prune -v --list $BDEST --keep-daily=7   # keep only the latest backup
                                             #    from each of the last 7 days
```

  - potentially very dangerous, as it will remove older copies
  - if you want to use borg as a time machine to restore old files, then you probably do not want to run `borg prune`
  - a good practice is to separate older archives by increasingly sparse intervals:

```
borg prune -v --list $BDEST --keep-daily=7 --keep-weekly=4 --keep-monthly=6
   # keep only the latest backup from each of the last 7 days
   #    + the latest from each of the past 4 weeks
   #    + the latest from each of the past 6 months
```

Step-by-step look at prune

```
/bin/rm -rf ~/tmp/{workspace,backups}/* ~/.config/borg/keys/*tmp_backups
export BORG_PASSPHRASE='justForThisDemo'    # probably bad idea; interactive much safer
borg init --encryption=keyfile ~/tmp/backups

cd ~/tmp/workspace
echo hello > a.txt
echo hello > b.txt
borg create --stats --list --filter='AM' ~/tmp/backups::monday ~/tmp/workspace
/bin/rm b.txt
echo hello > c.txt
borg create --stats --list --filter='AM' ~/tmp/backups::tuesday ~/tmp/workspace

borg list ~/tmp/backups             # shows both archives
borg list ~/tmp/backups::monday     # a,b
borg list ~/tmp/backups::tuesday    # a,c

borg prune -v --list ~/tmp/backups --keep-last=1   # keep only the last archive

borg list ~/tmp/backups             # only last archive
borg list ~/tmp/backups::monday     # does not exist!
borg list ~/tmp/backups::tuesday    # a,c

cd ~/tmp/restore
borg extract ~/tmp/backups::tuesday   # extract a, c
```

Remote backups over SSH

Two distinct modes:

1. Local BORG can access the remote system via SSH (slow)
   - in all examples, simply replace the local path `~/tmp/backups` with a remote path `user@name.domain.ca:/path/to/backups`
2. Use local BORG as a client and remote BORG as a server (much faster)
   - on the remote system, install borg via `` `pip` `` into your own Python 3 virtual environment (no need for root access)
   - the key file will be stored on your local system

Remote backups over SSH (cont.)

```
BDEST="user@name.domain.ca:/path/to/backups"
BPATH="--remote-path=/path/to/borg-env/bin/borg"
BFLAGS="--stats␣--list␣--filter='AM'␣--compression=lz4"
BPRUNE="--keep-daily=7␣--keep-weekly=4␣--keep-monthly=6"

# borg      init --encryption=keyfile $BDEST              # local borg
borg $BPATH init --encryption=keyfile $BDEST              # client-server

name=$(date "+%Y%b%d%H%M")    # e.g. 2021Feb091256
# borg      create $BFLAGS $BDEST::$name ~/tmp/workspace  # local borg
borg $BPATH create $BFLAGS $BDEST::$name ~/tmp/workspace  # client-server

# borg      prune -v --list $BDEST $BPRUNE                # local borg
borg $BPATH prune -v --list $BDEST $BPRUNE                # client-server
```

Demo: putting it all together

1. Remove all previous DAR backups

```
/bin/rm -f /Volumes/gdrive/test001/boa*
```

2. Nuke the BORG repo

```
/bin/rm -rf /Volumes/gdrive/test002/* ~/.config/borg/keys/*_test002
```

3. Initialize a new BORG repo and store the key

```
borg init --encryption=keyfile /Volumes/gdrive/test002
borg key export /Volumes/gdrive/test002 /some/safe/location
```

4. Explore (pay attention to the remote options)

```
source /path/to/functions202102.sh
e functions202102.sh       # open in the text editor
backup
...
```

Restoring

- Restoring DAR backup (might need a password)

```
for file in /Volumes/gdrive/test001/boa{0..99}; do
    if [ -f $file.1.dar ]; then
        echo --- from $file:
        dar -R ~/tmp/restore -O -w -x $file -v -g tmp/workspace \
                -g Documents/notes -g Documents/projects -g Desktop
    fi
done
```

- Restoring BORG backup (need both the key and the password)

```
repo=/Volumes/gdrive/test002
# borg key import $repo /some/safe/location/keyFile   # if need to restore the key
borg list $repo      # use the latest archive in the next line
latest=2021Feb092145
borg list $repo::$latest Users/razoumov/Documents/notes   # list files
cd ~/tmp/restore
borg extract $repo::$latest Users/razoumov/{tmp/workspace,Documents/notes,\
       Documents/projects,Desktop}
```

Summary

- Today we covered DAR (modern-day replacement to TAR) and BORG (deduplicating backup program)
  - DAR 's archived files are per backup (need to restore all previous backups)
  - BORG 's archived files are cumulative (appear in the latest backup unless previously deleted)
- Don't go crazy ... simple weekly rotation between methods and drives and local/remote should be sufficient
  - just remember your most recent destination!
- Store the passwords and exported keys in a safe place
- Prune with caution
- With all today's scripts and workflows, please exercise common sense
  - obviously, modify the scripts to suit your needs
  - make sure you understand what you are doing: don't use these scripts as black boxes
  - when running `backup()`, using verbose flags and monitoring live progress really helps
  - restore periodically to test things, e.g. use backups as historical archive

Questions?