$\log_2$rotate 1.0

Chris Forno (jekor)

August 20th, 2009

1 Introduction

$\log_2$rotate is designed to rotate backups with an optimal balance between retention and space usage. Instead of rotating backups using some familiar method such as daily, weekly, monthly, and yearly periods, it rotates backups using exponentially-growing periods. The exponential periods are based on the base 2 logarithm ($\log_2 x$) or squaring (x^2), depending on how you look at it.

$\log_2$rotate makes the guarantee that the distance between the nth and (n+1)th backups will be no greater than twice the distance between the (n-1)th and nth backups. The optimal is periodically achieved. For example, on the 64th rotation, $\log_2$rotate will recommend retaining the backups numbered: 1, 2, 4, 8, 16, 32, and 64.

Table 1: Backup periods for a daily-weekly-monthly-yearly schedule

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Day	1	2	3	4	5	6	7	14	28	35	65	95	125	155	185	215	245	275	305	335	365

Table 2: Backup periods for a $\log_2$rotate schedule

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Day	1	2	3	5	9	13	21	29	45	77	109	173	237	365

With a year's worth of backups, the $\log_2$rotate schedule saves 50% over the daily-weekly-monthly-yearly schedule.

The number of backups required for a given number of backup periods n will always be in the range $[\log_2 n, 2\log_2 n)$. For example, storing 5 years of backups will require 21 or fewer backups (with a best case of 11 backups). Storing 50 years of backups will require 28 or fewer backups, etc.

2 Algorithm

Let's number backups from newest to oldest, beginning with 1. We can define a function that given the number of backups made to-date, will return the set of backups that we should keep.

Let $B(n)$ be the backup function and n be the number of backups performed to date.

$$B(n) = \begin{cases} 1 & \text{if } n = 1 \\ n + B(n - 2^{\lfloor \log_2 n \rfloor - 1}) & \textit{otherwise} \end{cases} \qquad \text{for } n \in \mathbb{N}^*$$

Table 3: the algorithm in action

Day															
1	1														
2	1	2													
3	1	2	3												
4	1	2	3	4											
	↓	↓		↓											
4	1	2	·	4											
5	1	2	3	·	5										
6	1	2	3	4	·	6									
	↓	↓		↓		↓									
6	1	2	·	4	·	6									
7	1	2	3	·	5	·	7								
8	1	2	3	4	·	6	·	8							
	↓	↓		↓				↓							
8	1	2	·	4	·	·	·	8							
9	1	2	3	·	5	·	·	·	9						
10	1	2	3	4	·	6	·	·	·	10					
	↓	↓		↓		↓				↓					
10	1	2	·	4	·	6	·	·	·	10					
11	1	2	3	·	5	·	7	·	·	·	11				
12	1	2	3	4	·	6	·	8	·	·	·	12			
	↓	↓		↓				↓				↓			
12	1	2	·	4	·	·	·	8	·	·	·	12			
13	1	2	3	·	5	·	·	·	9	·	·	·	13		
14	1	2	3	4	·	6	·	·	·	10	·	·	·	14	
	↓	↓		↓		↓				↓				↓	
14	1	2	·	4	·	6	·	·	·	10	·	·	·	14	
15	1	2	3	·	5	·	7	·	·	·	11	·	·	·	15
16	1	2	3	4	·	6	·	8	·	·	·	12	·	·	·	16
	↓	↓		↓				↓								↓
16	1	2	·	4	·	·	·	8	·	·	·	·	·	·	·	16

3 Usage

log$_2$rotate follows the Unix philosophy. It does not rotate backups directly. It merely makes recommendations on which backups to keep. It's up to you to do the rest.

For simplicity, log$_2$rotate assumes that backups are made daily. The dates of the backups are read via stdin, in ISO 8601 format, delimited by whitespace. Recommended dates are output also in ISO 8601 format and separated by spaces.

4 Example

I originally created log$_2$rotate to rotate database backups for a web application. Each night, a backup of the database is created and sent to Amazon S3. After the backup succeeds, I run a backup rotation script that makes use of the `log2rotate` command. Here is the script, edited for simplicity (it assumes that the bucket contains nothing but the backup files):

```
#!/bin/bash

# List the backup dates and delete the ones log2rotate suggests we get rid of.
# s3cmd ls will give the date of the files in ISO 8601 format in the first 10
```

```
# characters of each line.
for f in `s3cmd ls s3://some-bucket/ | cut -c 1-10 | log2rotate --delete`; do
    s3cmd del s3://some-bucket/$f
done
```

Note the use of the `--delete` switch. You can instruct log$_2$rotate to output either deletion or retention (`--keep`) recommendations, depending on your needs.

5 Recommendation Failures

The algorithm works on the assumption that its recommendations have been followed the previous day. log$_2$rotate will check the backup set against that assumption and exit if it's not met (with the message "Recommendation failure!"). We do this for safety, because deleting backups from a set in an unexpected state will mean that the set might no longer meets retention guarantees.

If you'd like to proceed anyway, pass the `--unsafe` switch. This switch must also be passed on the first use of log$_2$rotate if you already have existing daily backups. Beware that I have only tested rotating an existing set of daily backups. I have not tested rotating any existing set of backups that was already following a different rotation pattern (such as daily-weekly-monthly-yearly).

6 Implementation

log$_2$rotate is a Haskell program. It has been tested with GHC 6.10.

Here are some necessary imports.

```
import Data.List (sort, (\\), genericTake)
import Data.Maybe (catMaybes)
import Data.Time.Calendar (Day, diffDays, addDays)
import Data.Time.Format (ParseTime, parseTime)
import System.Console.GetOpt (getOpt, OptDescr (..),
                              ArgDescr (..), ArgOrder (..), usageInfo)
import System.Environment (getArgs)
import System.Exit (exitWith, ExitCode (..))
import System.IO (stderr, hPutStrLn)
import System.Locale (defaultTimeLocale)
```

backupsToKeep is the core of the program.

```
backupsToKeep :: Int → [Int]
backupsToKeep n
  | n ≡ 0      = []
  | n ≡ 1      = [1]
  | n > 1      = n : backupsToKeep (n − (2 ↑ (floor (logBase 2 (fromIntegral n)) − 1)))
  | otherwise = ⊥
```

But *backupsToKeep* is just an idealized function over natural numbers. More often we're going to be working with dates/times. To keep things simple for now, we'll assume daily backups and will deal with only dates. To make things even easier, we'll work with just ISO 8601 formatting.

daysToKeep and *daysToDelete* correspond with the `--keep` and `--delete` switches. Both of these are actually the `--unsafe` version because they don't check the backup set against expectations, they just fill in the days with *fillInDays*.

```
daysToKeep :: [Day] → [Day]
daysToKeep ds = map (λi → days !! (i − 1)) (backupsToKeep $ length days)
  where days = reverse $ fillInDays ds


daysToDelete :: [Day] → [Day]
daysToDelete days = days \\ daysToKeep days
```

This takes a list of dates and fills in all of the gaps, giving you a contiguous list of dates.

```
fillInDays :: [Day] → [Day]
fillInDays []     = []
fillInDays days = genericTake (diffDays (last days) (head days) + 1) $
                  iterate (addDays 1) (head days)
```

If the user has not specified `--unsafe`, we'll only make recommendations if the backup set meets our expectations (it looks as if our recommendations were followed last rotation cycle).

To do so, simply remove the newest item from the list and check if it conforms to the $\log_2$ pattern.

```
gapless :: [Day] → Bool
gapless []  = True
gapless [_] = True
gapless xs  = diffDays (last xs) (last $ init xs) ≡ 1 ∧
              daysToKeep (init xs) ≡ init xs


safeRecommendation :: ([Day] → [Day]) → [Day] → Maybe [Day]
safeRecommendation f days = if gapless days
  then Just $ f days
  else Nothing


parseISO8601Date :: ParseTime t ⇒ String → Maybe t
parseISO8601Date = parseTime defaultTimeLocale "%F"


parseDays :: String → [Day]
parseDays = catMaybes ∘ map parseISO8601Date ∘ words


version :: String
version = "1.0"


usage :: String
usage = "Usage: log2rotate -delete|-keep [option...]"


data Flag = Help | Version | Delete | Keep | Unsafe
          deriving (Show, Eq)
```

```
options :: [OptDescr Flag]
options =
  [Option ['h'] ["help"]    (NoArg Help)    "show help",
   Option ['v'] ["version"] (NoArg Version) "show version",
   Option ['d'] ["delete"]  (NoArg Delete)  "show dates to delete",
   Option ['k'] ["keep"]    (NoArg Keep)    "show dates to keep",
   Option ['u'] ["unsafe"]  (NoArg Unsafe)  "make unsafe recommendations"]


log2rotateOptions :: [String] -> IO [Flag]
log2rotateOptions argv =
  case getOpt Permute options argv of
    ([],     _, [])   ->  ioError (userError ("\n" ++ showUsage))
    (o,      _, [])   ->
      if Help ∈ o
        then putStrLn showUsage >> exitWith ExitSuccess
        else if Version ∈ o
          then putStrLn (showVersion ++ "\n") >> exitWith ExitSuccess
          else if (Delete ∈ o ∧ Keep ∈ o) ∨ ¬ (Delete ∈ o ∨ Keep ∈ o)
            then putStrLn ("You must specify that you want either " ++
                           "retention or deletion suggestions\n" ++
                           showUsage) >>
                 exitWith (ExitFailure 1)
            else return o
    (_,      _, errs) ->  ioError (userError (concat errs ++ showUsage))
    where showVersion = "log2rotate version " ++ version
          usageHeader = unlines [showVersion, usage]
          showUsage   = usageInfo usageHeader options


main :: IO ()
main = do
  opts ← getArgs >>= log2rotateOptions
  let f = if Delete ∈ opts
    then if Unsafe ∈ opts
      then Just ∘ daysToDelete
      else safeRecommendation daysToDelete
    else if Unsafe ∈ opts
      then Just ∘ daysToKeep
      else safeRecommendation daysToKeep
  getContents >>= maybe (hPutStrLn stderr "Recommendation failure!" >>
                         exitWith (ExitFailure 2))
                        (putStrLn ∘ unwords ∘ map show) ∘ f ∘ sort ∘ parseDays
```