Pulling from SHEDS Database

Daniel Hocking

May 12, 2015

Keep Passwords Protected: .RProfile

To avoid saving the password in any scripts use an option variable in R, which is sort of like an environmental variable in the OS, but is just within R.

All you need to do is:

1. Create a file named `~/.Rprofile` (if it doesn't already exist) in your home directory on your local computer (unless your using the rstudio server on felek or osensei, then do it in your home directory on that server). This file gets loaded automatically every time you start R.
2. Within `~/.Rprofile`, set two option variables called, say, `SHEDS_USERNAME` and `SHEDS_PASSWORD` using the `options()` function:

```
options(SHEDS_USERNAME='<user>')
options(SHEDS_PASSWORD='<password>')
```

just replacing `<user>` and `<password>` with your specific username and password.

The one exception to putting it in the root directory is when using `packrat` for managing package dependencies. If using packrat in RStudio you need to put the .RProfile in the project's directory then `source` the file at the beginning of your R script. A description can be found at http://rstudio.github.io/packrat/limitations.html

Simple Queries from Small Tables: dplyr

In R using dplyr and refering to the username and password stored in your R profile you can connect to the database, query, and fetch (collect) the results. `dplyr` is a nice package that will convert the `dplyr` R code into SQL.

```
db <- src_postgres(dbname='sheds',
                   host='felek.cns.umass.edu',
                   port='5432',
                   user=options('SHEDS_USERNAME'),
                   password=options('SHEDS_PASSWORD'))
```

then test if this works:

```
tbl_agencies <- tbl(db, 'agencies')
```

To view the SQL query that was run from this you can run `tbl_agencies$query`. If you are better with dplyr code than SQL this can be useful for getting some SQL examples, although they are not always the most efficient (pretty amazing and useful translator either way!).

Then you can get (fetch/collect) the results of the query to store them in a dataframe in R using

```
df_agencies <- collect(tbl_agencies)
```

This was just a very simple example of getting the whole table. It is easy to add filters, joins, aggregations, etc. using dplyr. An example for getting all the observed temperature data:

```
db <- src_postgres(dbname='sheds', host='felek.cns.umass.edu', port='5432', user=options('SHEDS_USERNAM

# table references
tbl_locations <- tbl(db, 'locations') %>%
  rename(location_id=id, location_name=name, location_description=description) %>%
  select(-created_at, -updated_at)
tbl_agencies <- tbl(db, 'agencies') %>%
  rename(agency_id=id, agency_name=name) %>%
  select(-created_at, -updated_at)
tbl_series <- tbl(db, 'series') %>%
  rename(series_id=id) %>%
  select(-created_at, -updated_at)
tbl_variables <- tbl(db, 'variables') %>%
  rename(variable_id=id, variable_name=name, variable_description=description) %>%
  select(-created_at, -updated_at)
tbl_values <- tbl(db, 'values') %>%
  rename(value_id=id)
tbl_daymet <- tbl(db, 'daymet')
tbl_covariates <- tbl(db, 'covariates')

# list of agencies to keep
# keep_agencies <- c('MADEP', 'MAUSGS')

# fetch locations
df_locations <- left_join(tbl_locations, tbl_agencies, by=c('agency_id'='agency_id')) %>%
  # filter(agency_name %in% keep_agencies) %>%
  filter(agency_name != "TEST") %>%
  rename(featureid=catchment_id) %>%
  collect
summary(df_locations)
unique(df_locations$agency_name)

# fetch temperature data
df_values <- tbl_values %>%
  left_join(tbl_series, by = c("series_id")) %>%
  left_join(dplyr::select(tbl_variables, variable_id, variable_name),
            by=c('variable_id'='variable_id')) %>%
  dplyr::select(-file_id) %>%
  filter(location_id %in% df_locations$location_id,
         variable_name=="TEMP") %>%
  collect %>%
  mutate(datetime=with_tz(datetime, tzone='EST'),
         date = as.Date(datetime),
         series_id = as.character(series_id)) %>%
  rename(temp = value)
summary(df_values)
```

Complex, Small-Medium Sized Queries: RPostgreSQL

Sometimes dply isn't going to have the most efficient SQL tanslation or you want to do something more complex or use special postgres functions. Here is an example using the `RPostgreSQL` package (which dplyr relies on) directly. This requires writing your own SQL code and in this example we use a special spatial join in postgres to determine which HUC8 each catchment centroid is located within.

```
#-------------------------Get HUC8 & HUC12 & add to covariate df----------------
# pass the db$con from dplyr as the connection to RPostgreSQL::dbSendQuery
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname='sheds', host='ecosheds.org', port='5432', user=options('SHEDS_USERNAME'),
rs <- dbSendQuery(con, "SELECT c.featureid as featureid, w.huc8 as huc8
FROM catchments c
JOIN wbdhu8 w
ON ST_Contains(w.geom, ST_Centroid(c.geom));")

# fetch results
featureid_huc8 <- fetch(rs, n=-1)
```

Complex, Large (Any) Queries: psql command line

When doing queries that take a long time, R with hang and RStudio will lose the connection. To avoid this problem, when smaller queries cannot be done in a loop (see above), it's best to do the query directly via the command line. In the case of the SHEDS database, this occurs most frequently when trying to query the `daymet` table, especially if trying to get a summary metric for all featureid (e.g. mean annual precipitation for all catchments).

To learn PostgreSQL the official tutorial is a good place to start: http://www.postgresql.org/docs/9.3/static/tutorial.html

I also recommend attending a 2-day Software Carpentry workshop. They cover automating taksing using the Unix shell, structured programming in Python, R, or MATLAB, and version control using Git or Mercurial. Some bootcamp workshops also cover relational databases. They have a brief tutorial on relational databases at http://swcarpentry.github.io/sql-novice-survey/.

Create New Database then Import

The best way to do handle queries that involve large tables or complex queries would probably be to create a new table in the database by running the SQL within psql directly. Once that's done, then you could just pull the results directly into R.

Previous problems with connectivity were likely a result of trying to run a long query and simultaneously export the results (whether thats to a csv file or piping directly back to R). So it should be more robust to run the query, save the result in the database as a new table, and then fetch the rows from the new table into R when its done.

For example to get the mean annual precipitation, create an sql file `mean_annual_prcp.sql` that contains:

```
CREATE TABLE daymet_annual_prcp AS (
  SELECT featureid,
         extract(year from date) as year,
         avg(prcp) as prcp
  FROM daymet
```

```
  WHERE featureid IN (34234,234234,...)
  GROUP BY featureid, year;
)
```

This SQL script can be created in R so as to link the list of desired featureid:

```
unique_id <- unique(df$featureid)
unique_id_string <- paste(unique_id, collapse=', ')

qry <- paste0("
              CREATE TABLE annual_prcp AS (
                SELECT featureid,
                       extract(year from date) as year,
                       avg(prcp) as prcp
                FROM daymet
                WHERE featureid IN (", unique_id_string, ")
                GROUP BY featureid, year;
                )"
              )

write.table(qry, "Code/mean_annual_prcp.sql")
```

Then ssh into the server run:

```
$ psql -d sheds -f Code/mean_annual_prcp.sql
```

When that is done. Go back into R and run

```
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname='sheds', host='ecosheds.org', port='5432', user=options('SHEDS_USERNAME'),
rs <- dbSendQuery(con, "SELECT *
                        FROM annual_prcp;")

# fetch results
df_annual_prcp <- fetch(rs, n=-1)
```

Problem with connectivity

In the cases of queries that take a long time, it's likely that the connection to the database will be lost and only part of the query will be returned. GNU Screen handles this automatically so that might be the easiest option: http://en.wikipedia.org/wiki/GNU_Screen

Another option is to add an `&` to the end of the psql command when using the unix shell (Terminal on a Mac) so that it runs in the background.

```
$ psql -d sheds -f Code/mean_annual_prcp.sql > Data/annual_prcp_results.csv &
```

However, running in the background prevents psql from asking for your password. You can use a pgpass file: http://www.postgresql.org/docs/9.3/static/libpq-pgpass.html to get around this.

The idea here is that you just save your password in this file, but only you (and root) can access it, so it's fairly secure. Then whenever you run psql to connect to the database, psql will automatically look for this file in your home directory, and if it exists use the password from there rather than prompting you.

All you need to do is create a file: `~/.pgpass` (so its in your official home directory at `/home/<user>/.pgpass`)

Within that file, have one line that looks like this:

```
localhost:5432:*:<user>:<password>
```

Then you need to change permissions on that file so that only you (user) can read/write to it (if you don't do this, postgres will complain about it not being secure). So to change the permissions to rw for the user only, run:

```
$ chmod 600 ~.pgpass
```

Then running psql to connect to `sheds`

```
$ psql conte_dev
```

If the pgpass file worked, then it shouldn't ask for your password. To see the jobs running (including in the background) enter:

```
$ jobs
```

If you exit the ssh and then log back in, you won't be able to see it after running `$ jobs` (because you've opened a new session), but if you run:

```
$ ps -u <user>
```

then you should see it listed there (this lists all processes for your user, not just current session). *Be careful doing this though because even if you kill the job, the postgres query might continue running, and would need to be killed from within postgres.*

Putting it all together

The only problem with the current system is that you have to go back and forth between R and the shell.

make

shell script

Problem: automating ssh connection to do psql query???