



## ETL Nightmares

Thomas MacKenzie  
thomas\_mackenzie78@yahoo.com

### Introduction

Moving data from one place to another often seems like a trivial task. The scripts to handle this start off simple, but because they perform a useful task, they become critical to the business. Sticking with the same thing you started with usually creates headaches and leads to frustration.

Although shell scripts seemed like a good place to start, Perl can do all that better and faster. Rethinking the problem, judicious use of data structures, and consistency across the entire product will help you sleep better at night and avoid the nightmares I've run into.

### ETL Basics

ETL is an acronym for, "Extract, Transform, and Load". A plain vanilla ETL process extracts from some data source, transforms it, and then loads it into another data store. The source of the data could be anything: an internal database, a foreign database, or even a file. The transformation converts the source data from its original form into that required by the destination. Once the data is transformed, it is then loaded into the target data store. This process can be run manually or scheduled to run unattended at specified times. ETLs replace tedious data entry tasks and save tremendous amounts of time when done correctly. When designed poorly, they overwhelm support efforts and give Perl a bad a reputation.

### What do I know?

I have spent the better half of my career as a consultant reengineering ETL processes. Most developers cringe when they hear someone like me is around. My purpose is to dissect the ETL processes and the databases they have shed blood,

sweat, and tears to develop. Once I figure out how everything works I reengineer their system with sound design patterns and clean Perl. I have seen processes that go against even the basics of good software engineering.

I separate these problems into two categories: bad practices and bad code. Bad practices include not using a versioning system (or even keeping backups), using the shell for complex ETLs, and not using a standard naming convention throughout the process. The bad code is some of the worst you may ever see; not using strict, confusing and inconsistent data structures, statements that are several line lengths long, and worst of all, commentless code.

### Back that script up!

I have evaluated ETL processes where no scripts were put into a versioning system; not even making a back up copy! Small side projects can often grow into critical processes. What starts as a quick and dirty one-off script often becomes a useful tool. Somewhere during this transition the developer does not back up the code with a versioning tool. This is the primary point of the decline to bad practices. A versioning system is crucial on many levels. Once the process depends on a script, you need to be able to track its changes and recover it if a machine crashes. There is often no provision for recovery so if the one production machine has a catastrophic failure you lose all of the code. You only need to have a hard disk crash on you once to never want to go through that pain again.

Does this mean you should dump all projects into a versioning system? Absolutely. My rule is that anything valuable (or that could be valuable later) to anyone down the road needs a back up copy. Come on, disks are cheap!