Some Simple Guidelines for Effective Data Management

Data are at the heart of empirically based sciences, serving as the primary evidence supporting (or refuting) models of the way that our natural world operates. While most scientists spend much time thinking about the types of data they need to further their studies, and often collect and analyze their data using sophisticated techniques and approaches, relatively little effort is spent considering how to store their data. An unfortunate result of this cursory data management is that much valuable information gets lost. This is a waste when one recognizes that scientific observations capture the values and relationships among an interesting set of objects or processes at some place and time, and typically contain information useful for investigating research questions that were never anticipated when the data were originally collected. Many ecological studies can benefit from access to additional data with similar thematic foci, or collected from the same spatial region during the same time period. Synthetic, integrative research in ecology in particular depends on using data typically collected by multiple parties that were not coordinated in their research approach. For these reasons, it is increasingly important to store and document scientific data in ways that facilitate their effective retrieval and interpretation in the future.

This document provides some simple guidelines for effective data management, which, if put into practice, will benefit the original data owner as well as enhance prospects for the long-term preservation and re-use of the data by other researchers. The rules of thumb presented here facilitate rapid and accurate interpretation of the data, and pre-dispose the data to more effective processing by computers. We hope that you will embrace these simple rules, which are based on general data-modeling principles that are broadly relevant to any types of information that you might collect.

1. Use a scripted program for analysis.

While it's a pain to start—since you are learning a whole new language—using a scripted analysis program, such as the R statistical package (which is free, flexible, and *very* good), MATLAB, or SAS will save you countless headaches in the future. Analysis scripts are written records of the various steps involved in processing and analyzing data, and provide a form of analytical "metadata." Such scripts can be easily revised and re-executed any time you need to modify an analysis, which is frequent after comments from advisors, committee members, and reviewers. This scripted approach is in contrast to a "GUI-driven" analysis, in which various changes to data are made by selecting and altering values in place, and by choosing various processing steps from drop-down menus, followed by a "run" or "execute" or "ok" button. Such GUI-based analyses often seem convenient when working through your data and analysis, but rarely leave a clear accounting of exactly what you have done. In contrast, when you use a scripted program for your analysis, you will always have a record of what you did with your data from the time you collected it to the time you publish it, so it's easy to recollect your decisions, even after a few years have passed.

2. Store data in nonproprietary software formats (e.g., comma delimited text file, .csv); proprietary software (e.g., Excel, Access) can become unavailable, whereas text files can always be read.

Excel and Access have not always been the preferred spreadsheet/data storage programs, and in the future they will, undoubtedly, be replaced by others—or newer, potentially incompatible versions. If your data files are stored using proprietary software, when this software is no longer available, your data also will disappear. In general, what this means is that in a decade or two, you may not be able to open your hard-won and precious data files. (For example, none of us can now digitally access our undergraduate thesis data or documents.)

3. Store data in nonproprietary hardware formats.

The same dictum for software is also true for hardware – formats can rapidly become obsolete. We know many scientists who have valuable data that are essentially lost because they are stranded on old formats, such as 8-track tapes, or 5.25" floppy disks. In this case, a best "nonproprietary format" would be the Internet itself, rather than various archival media. As hard as it is to believe today, we can foresee the day when CD-ROMs might be difficult to read. But the continuity of the Internet seems more assured. The Ecological Society of America encourages authors to archive proofed and documented data sets ‹http://www.esa.org/science_resources/datasharing.php› and provide one long-term data storage solution, Ecological Archives ‹http://esapubs.org/archive/default.htm›. At various times, it is also advisable to create additional copies of your data that are off-line (not on the Internet), using the most popular medium of the day. As of 2008, this is probably the DVD, although it has a limited capacity (~8 GB) for some purposes.

4. Always store an uncorrected data file with all its bumps and warts. Do not make any corrections to this file; make corrections within a scripted language.

When you make corrections post facto to an original data file you could easily be changing something that you later discover was correct in its original form, or maybe you make a mistake while correcting

another mistake. Using a scripted language, you can re-run analyses as well as transformations and corrections to your data, by using your original data as input, but saving the changes to a separate data file. This way you always have easy access to your original data values. Documenting your scripted code with "comments" can further clarify why you might be changing certain values. By saving the original data set with copious notes, you can maintain your data integrity for the long term. Consider making your original data file read-only, so it cannot be inadvertently altered.

5. Use descriptive names for your data files.

File names (or table names) are still the easiest way to indicate the contents of any given data file. So it is useful to try to give your data files names that are terse, but indicative of their content. Thus, rather than calling a file "Year1" or "Summer07", you might call it something like "Corvallis_VegBiodiv_2007". Generally, capturing some hints of the place, time, and theme can be extremely useful in the file name, even if done in a highly abbreviated manner. There are two caveats about this approach. First, do not make your file names overly long; extremely long names will make it more difficult to identify and import files into your analytical scripts. Second, do not use blank spaces in your file names, since some applications have trouble importing file names with blank spaces in them. Instead use "CamelCaps" to indicate names created by stringing words together. This is where you string words together, but capitalize each word to differentiate it. The "_" (underscore) or "-" hyphen symbols can also be used to separate distinct parts of a name, as in the example provided.

6. Include a "header" line that describes the variables as the first line in the table.

A standard convention that many software applications understand is that the first line (or record) of a table is the "header" line, which lists the names of the variables in the file. Subsequent lines are then interpreted as data. Similar to the standards for naming files as described above, variable names should be terse but descriptive. Also, these names should not include blank spaces because many applications cannot easily process variable names with blank spaces in them (e.g., below you might use "Sp1" or "Sp_1" instead of "Sp. 1"). Many applications can automatically use "header" line information to identify and name the columns in a table, as distinct from the values within the columns. As a corollary, choose your variable names carefully to help indicate what values they are holding. For example, "Sp1" might be better named "Sp1_Count". Under section No. 8 we will see that there are even better ways of structuring the data from this particular example.

Site	Sp1	Sp2	Sp3	Sp4	Sp5
XYZ	12	4	8	16	3

7. Use plain ASCII text for your file names, variable names, and data values.

It is unfortunate that in much of today's software, diacritical marks commonly used in foreign languages such as Spanish or French (e.g., accents or tildes), or ideograms (e.g., Chinese or Japanese characters), are not well supported. So it is wise to avoid these where possible at this time, by trying to

use only ASCII characters, which are based on the English alphabet, for file names, variable names, and data values. The ASCII code includes all the English letters (uppercase and lowercase) and numbers, and many common punctuation marks. Note that only letters, numbers, hyphens ("-"), and underscores ("_") should be used in file names and variable names, but data values can include blank spaces, punctuation, etc. However, one should avoid using certain specialized ASCII characters in data values, since these can lead to unexpected results. For example, one can encode ASCII values for "tabs," "backspace," and "carriage return" into your data values, but don't do this unless absolutely necessary, and you are sure your processing software can properly interpret this information.

8. When you add data to a database, try not to add columns; rather, design your tables so that you add only rows.

When analyzing data using a scripted language (e.g., R, SAS, MATLAB), you will typically read in your data sets by column (site, date, …). The analysis program will expect that all rows contain comparable pieces of information (a list of sites, dates, or species abundances). If data sets can be compared among years or sites, etc., or can be analyzed in the same way, setting up files to maximize the consistency of column content will streamline future analyses.

Most databases or analysis software have an implied loop in which all rows of your data can be automatically analyzed as a set, but new columns (variables) will need to be explicitly added to code. For example consider (see below) how you would compute some average abundance across species and years, if the species' abundance information is in columns rather than rows. In this case, the data values across years are clearly meant to be comparable, yet you would have to examine the columns to see which are values for species' abundances, and then compute the average of the values in those columns. In each case, the code would have to be modified to account for any new columns of species' abundance data (Sp8, Sp9, etc.) that are added. The alternative, preferred structure in our example would only require three columns—to accommodate any number of species without needing to change the scripting code describing and averaging the data file.

An example of poor data management for the long term.

Corvallis_VegBiodiv_2006.csv

Site	Sp1	Sp2	Sp3	Sp4	Sp5
XYZ	12	4	8	16	3
PDQ	4	14	0	9	64

Corvallis_VegBiodiv_2007.csv

Site	Sp1	Sp2	Sp3	Sp4	Sp5	Sp6	Sp7
XYZ	21	0	0	45	0	12	6
PDQ	0	4	0	0	16	3	0

Example of effective data management for the long-term:

Corvallis_VegBiodiv_2006.csv

Site	SpName	Abundance
XYZ	Sp. 1	12
XYZ	Sp. 2	4
XYZ	Sp. 3	8
XYZ	Sp. 4	16
XYZ	Sp. 5	3
PDQ	Sp. 1	4
PDQ	Sp. 2	14
PDQ	Sp. 4	9
PDQ	Sp. 5	64

Corvallis_VegBiodiv_2007.csv

Site	SpName	Abundance
XYZ	Sp. 1	21
XYZ	Sp. 4	45
XYZ	Sp. 6	12
XYZ	Sp. 7	6
PDQ	Sp. 2	4
PDQ	Sp. 5	16
PDQ	Sp. 6	3

Even better would be to create a column where critical information, such as the year, is included as a column within the data table, and not just as metadata embedded in the file name.

Corvallis_VegBiodiv_2007.csv

Date	Site	SpName	Abundance
2007	XYZ	Sp. 1	21
2007	XYZ	Sp. 4	45
etc…			

9. All cells within each column should contain only one type of information (i.e., either text, numerical, etc.).

Again, for analysis, this is important. If there is a block of data followed by a few lines of text notes and summary statistics followed by another block of data, for example, cells within a column will not have the same meaning. Common data types include text (alphanumeric strings of text), numeric, date/time, Boolean (also called Yes/No or True/False), and comments (for storing large quantities of text). Most analysis software cannot handle mixed types of data within a column, and for those that do not, it is much harder to detect errors in the data, such as when a site name is used rather than a site code in a column. In the example below, your analysis software might misinterpret the comment on line 3 as a Site name!

Corvallis_VegBiodiv_2006.csv

Site	Sp1	Sp2	Sp3	Sp4	Sp5
XYZ	12	4	8	16	3
Following data were collected by EB's students					
PDQ	4	14	0	9	64

10. Record a single piece of data (unique measurement) only once; separate information collected at different scales into different tables. In other words, create a relational database.

If you find yourself repeating or copying a piece of information many times within a spreadsheet, for example site-level information such as latitude or average annual rainfall, it becomes unclear to others looking at your data (and you, in a few years) whether you happened to measure the same rainfall on several sample dates, or whether this is a single observation copied several times (flat file). This means that in the future, your data set may be difficult to interpret by yourself or others.

Example of effective data management for the long term.

Site-info-VegBiodiv.csv

Site	Latitude	Longitude	AvgPrecipitation	DomPlantCommu

Abund-data-VegBiodiv_2007.csv

Date	Site	SpName	Abundance

Species-info-VegBiodiv.csv

SpName	Vert1-Invert0	Endo1-Ecto0	AvgMass	TrophicStatus

Uniquely identify records within a table using a key field; make sure that each unique record has a unique value associated with it. For relational databases, as in this example, tables can be linked through these unique key fields (e.g., key variable "Site" can link Site-info-VegBiodiv.csv with Abund-data-VegBiodiv_2007.csv, or key variable "SpName" can link Abund-data-VegBiodiv_2007.csv with Species-info-VegBiodiv.csv, for example). When analyzing your data, you can then merge these separate data files together, using your scripted program to link data tables using these key fields. In relational databases, this is called a 'join." This method of joining tables together via keys is the basis for a relational database.

Let's think about an example.

Consider a case where you measure the severity of a disease on three species at a number of different sites along with a number of characteristics of each site (e.g., precipitation and elevation) in a single data table (a flat-file design).

Plot	Precip	Elev	Sp1	Sp2	Sp3
San Miguel-a	12.3	22.4	Hi		Lo
San Miguel-b	12.3	22.4	Med		
San Miguel-c	12.3	22.4		Hi	Lo
Anacapa-a	4.6	3.7		Lo	
Anacapa-b	4.6	3.7	Med		
Catalina-a	8.5	52.5		Lo	Lo

Looking at the above table, which is only a small sample of the entire data set, it appears that there are (at least) six distinct study areas, each indicated by an island name, but the multiple records per island might indicate separate sites on each island. However, it looks as if there might only be one weather station per island, and perhaps the elevation is taken from near there, as evidenced by the consistency in the precipitation and elevation measures within a given site.

A relational database would more effectively capture the above information in two tables.

SITES TABLE:

Site_ID	Island_Name	Precip	Elev
1	San Miguel	12.3	22.4
2	Anacapa	4.6	3.7
3	Catalina	8.5	52.5

DISEASE TABLE:

Site_ID	Plot_ID	SpeciesName	DiseaseSeverity
1	a	Sp1	Hi
1	a	Sp3	Lo
1	b	Sp1	Med
1	c	Sp2	Hi
1	c	Sp3	Lo
2	a	Sp2	Lo
2	b	Sp1	Med
3	a	Sp2	Lo
3	b	Sp3	Lo

Note that in the relational design, we can use the **Site_ID** key to associate the island names, precipitation, and elevation information with the appropriate plots in the Disease table.

Here are some problems that can be addressed by capturing these data in a relational database, as opposed to the single table.

a) *Reduce wasted space*. Since most species don't occur at most sites, the flat-file table format contains many empty fields. Tables in the relational version do not contain these missing values.

b) *Avoid structural changes*. If you make a measurement on a fourth species at one site you will have to change your overall database table structure by adding an additional field (column) if you were using the flat file. The relational version would simply involve adding another record (row) to the Disease table.

c) *Reduce redundant data entry*. You have to enter the same island name and precipitation value many times in the flat file, increasing the data entry effort and likelihood of errors. You only enter this information once in the relational model.

d) *Make querying easier*. For example, tallying up disease prevalence within a site requires checking a potentially variable number of fields (Sp1, Sp2, Sp3, etc.) in the flat-file design, instead of just summarizing *across records* as you would in a well-modeled relational database.

e) *Reduce redundant information*. Using a flat-file design as above, if you sample 50 locations at a site, the island name, site name, and precipitation will appear 50 times. This leads to a waste of storage space and processing time.

f) *Reduce chances for inconsistent data*. If you collect new data at a site (e.g., better precipitation records become available), or discover data entry errors that need to be corrected, it is easy to end up with inconsistent data for each site in a flat-file design. This is because any change, such as correcting the spelling of a site name, must happen for every record where it occurs, instead of only once in a relational database. This problem is compounded if the site names are incorrect in

more than one flat-file table. You would need to locate each of these tables and make the change over multiple records, whereas with the relational model, changing the value in the one table (e.g., the Sites table) would automatically correct the site name spelling for any other tables referencing it.

g) *Disaggregate data*. Store unique pieces of information in separate columns. For example, if you collect data in a "plot" "at a "site," represent these as separate columns in your data table so that your sampling design is clear. You may, in the future, want to examine all samples together from the same plot, which is harder if you have stored the data encoded in a single cell (e.g., site = "San Miguel-a"). In contrast, having separate cells for each type of data (e.g., site = "San Miguel," plot = "a") will facilitate this type of search. Similarly, consider separating genus and species information, although one should use the full binomial name in the species field, which leads to some redundancy with the contents of the genus field (e.g., genus = "*Homo*"; species = "*Homo sapiens*").

11. Record full information about taxonomic names.

Over time, the names of taxa often are changed as their evolutionary relationships are clarified. The same taxonomic name can actually refer to two or more different concepts of a species. However, scientific names in ecological data are fixed as originally recorded, and so it is critical for long-term preservation to document which taxonomic descriptions were intended by each taxon name used in a data set. This becomes particularly important when comparing species information from data collected at different times, as the names used in the data sets can be ambiguous, which affects calculations of diversity and richness, among other issues. The best way to clarify a taxonomic name is to document the taxonomic authority you are using for the name. For example, *Homo sapiens* Linn. clarifies that the authority for this binomial is Linnaeus. Unfortunately, there are several formats to choose from for specifying taxonomic authority information, but any reference information is better than none.

12. Record full dates, using standardized formats.

Data collected in the United States often uses a date representation that is ambiguous because it is not clear about which order the year, month, and day fields are listed. For example, '02-03-04' could represent March 2, 2004, but in other countries would be interpreted more likely as March 4, 2002, or even February 3, 2004. The international standard is to format date values as '2004-03-02', where the year is listed using four digits, followed by the month and day (in order of most significant to least significant component). This order has the added benefit of allowing data records to be easily sorted in date-order. A good format for datetime is: YYYY-MM-DDThh:mm:ss. This is the format for dates and time promoted by the International Organization of Standards, as detailed in ISO 8601 (2004). If the date and time are expressed in UTC (Coordinated Universal Time, which is essentially the same as Greenwich Mean Time, GMT), it is standard to indicate this by appending a "Z" to the end of the date/time.

13. Always maintain effective metadata.

While you are entering this year's data, it is easy to think you will always remember what the site looked like, which sample you ran over with the truck, or just how you assigned a single value for the

"depth" of a pond. Do not underestimate your ability to forget details about a study! Life will get busy, you will return to your site in 5 years to re-sample or you will sit down to analyze 3 years of data, and you will realize just how many details you have forgotten. Important details include why you are collecting data, exact details of methods, all of the names of data and analysis files associated with the study, definitions for data and treatment codes, including missing value codes, and names, definitions, and the unit of measurement for each column in a data table. A metadata file associated with your study will allow you to record information so that you, or others interested in extending your work, can return to it. You might even consider using a metadata standard such as Ecological Metadata Language (EML) to help guide you in what metadata are important to include for your data set, as well as provide a good structure for capturing this information (Fegraus et al. 2005).

Conclusion

By following these simple guidelines, your data will be less prone to error, more efficiently structured for rapid analysis, and more readily interpretable for any future questions that they might help elucidate. Indeed, you will thank yourself if and when you revisit these data in the future to investigate some interesting long-term trends, or discover your "old" data might be critical to informing some larger, integrative study.

Literature cited

Fegraus, E., S. J. Andelman, M. B. Jones, and M. Schildhauer. 2005. Maximizing the value of ecological data with structured metadata: an introduction to Ecological Metadata Language (EML) and principles for metadata creation. ESA Bulletin 86(3):158-168.

ISO 8601. 2004. Technical Corrigendum 1–Data elements and interchange formats–Information interchange–Representation of dates and times. The International Organization for Standardization, Geneva, Switzerland.

Low-key resources

‹http://www.geekgirls.com/menu_databases.htm›
‹http://www.htmlgoodies.com/primers/database/article.php/3478051›

A specific design example is at:
‹http://www.geekgirls.com/databases_from_scratch_3.htm›

Elizabeth T. Borer and Eric W. Seabloom
Department of Zoology
Oregon State University
Corvallis, OR 97331
E-mail: borer@science.oregonstate.edu

Matthew B. Jones and Mark Schildhauer
National Center for Ecological Analysis and
    Synthesis
University of California Santa Barbara, CA
    93106