Assignment 1

Revision 1 (Jan 26, 2015)

Overview

In this assignment, you will write a set of UNIX shell scripts that reviews and manages the list of packages installed on a Linux system. I write this kind of code when I am setting up a new system, to see what software I had installed on the old system that I don't yet have on the new one, etc.

To get started, unpack the utility and sample data file `assignment1.tgz`:

```
$ tar -xzvf assignment1.tgz
```

Revision Log

Revision 2 (Jan 27, 2015)

- Described how to set `PKGMF`

Revision 1 (Jan 26, 2015)

- Updated due date
- Specified that pc-export should exclude self-dependencies
- Updated sample data files
- Described sample data files

The `join` command

This assignment will make heavy use of the join command, which takes two files and determines their common lines (or lines missing in one, etc). It understands multi-column text (defaults to space or tab separating fields; this can be changed with the `-t` command line option), and uses a *key field* to identify matching lines; by default, this is the first field. Two lines match if they have the same key field. Both files must already be sorted by key field.

For example, if we have the following file of student information:

```
fhr1861,563-85-7100,Gutierrez,Kimberly,1983-10-19
ix2328,751-11-9936,Bishop,Justin,1989-03-09
kzr030,859-56-8075,Greene,Heather,1982-10-31
oip338,502-77-6044,Evans,Johnny,1984-10-15
```

```
osvc918,143-65-8809,Carroll,Donald,1982-12-30
wk4142,169-32-8372,Knight,Kathryn,1983-04-05
wsb6112,467-34-1848,Carter,Betty,1984-06-11
xh7856,420-55-4383,Willis,Samuel,1985-11-05
xnpz6156,938-00-3333,Hall,Patricia,1991-08-03
zup105,486-39-4464,Lynch,Joan,1987-11-20
```

And this file of grades:

```
ix2328,94
oip338,73
wk4142,88
```

Then the following command will print the grades for the students along with their student information:

```
$ join -t, students.csv grades.csv
ix2328,751-11-9936,Bishop,Justin,1989-03-09,94
oip338,502-77-6044,Evans,Johnny,1984-10-15,73
wk4142,169-32-8372,Knight,Kathryn,1983-04-05,88
```

It pastes the two files together, so the new column introduced by the second file is at the end of each row of the first file. Only rows appearing in both files are printed; the `-a1` option will make it also include rows from file 1 that do not have a match in file 2 (they just won't have a grade column). There is an another option, `-v`, that makes it only print lines from one file that do *not* have a match in the other file. Finally, specifying `-` for one of the file names makes it read that file's data from standard input instead of a file.

Problem Description

Packages and Package Sets

On a Linux system, software is usually installed in *packages*; each package has a set of *dependencies*, which are other packages that it requires in order to function. Your scripts will operate on package sets, helping the administrator to identify missing and extra packages and compare the versions of packages installed on two systems.

Each package set will be described by two files:

`pset.versions` This file contains a list of the name and version, separated by a comma, of all packages in the set. Example:

```
ebtables,2.0.10-13.el7
nscd,2.17-55.el7_0.3
kpartx,0.4.9-66.el7
mailx,12.5-12.el7_0
harfbuzz,0.9.20-3.el7
gnupg2,2.0.22-3.el7
python-pip,1.5.6-5.el7
hicolor-icon-theme,0.12-7.el7
redhat-support-lib-python,0.9.6-0.el7
libXext,1.3.2-2.1.el7
rhn-client-tools,2.0.2-5.el7
```

`pset.deps` This file contains the dependencies of packages, again tab-separated, with one line for each dependency relationship. For example, the following line:

```
make,info
```

means that the `make` package requires the `info` package in order to function (`info` is a *dependency* of `make`).

A package set will have a name; for example, the package set `wanted` will be represented by the files `wanted.versions` and `wanted.deps`.

Support Code

I am providing you with 2 scripts to help with your work:

`pc-list-packages` This script lists the packages installed on the system; this is in the same format as used in the `.versions` file of a package set. You cannot make any assumptions about the order of its output.

`pc-get-deps` This script takes a single package name on the command line and prints out a list of its dependencies, one per line.

I am also providing some example package manifests that can be used by scripts. The scripts look for an environment variable, `PKGMF`, to find a package manifest to use. If `PKGMF` is set to the path to some package manifest file, such as `system1.mf`, then they will list packages and dependencies described in `system1.mf`. If `PKGMF` is not set, then they assume that you are running on a Red Hat or Fedora Linux system and try to query the system's native package manager.

You can set the variable like this:

```
export PKGMF=base.mf
```

You can also set it for an individual command:

```
PKGMF=base.mf ./pc-export-packages base
```

Requirements

Your job is to write four scripts.

`pc-export-packages`

The `pc-export-packages` script takes a single command-line parameter, the name of the package set, and produces `pset.versions` and `pset.deps` files for that package set. It gets the package data from `pc-list-packages` and `pc-get-deps`.

This file is intended to be run on a system to prepare a package set for analysis.

This command will need to first run `pc-list-packages` to list the packages, and then loop over the resulting list of packages and call `pc-get-deps` for each package and process its output appropriately.

For various reasons, packages on a system may depend on themselves. However, for the purposes of managing installed package sets, these self-dependencies do not matter. Therefore, `pc-export-packages` **must exclude** self-dependencies (for a package `foo`, the output line `foo,foo` must never occur).

This script should have no ordinary output.

Note: writing the following scripts will be easier if this script makes sure its output is always sorted.

`pc-leaves`

The `pc-leaves` script takes a single argument, the name of a package set.

It prints out all the *leaf packages*: those packages that are not the dependencies of any other package. In a package set `foo`, it is those packages that never appear in the second column of `foo.deps`.

Leaf packages are interesting because they represent the smallest set of packages about which an administrator needs to make a decision. If a package is used by another package, the admin doesn't really need to decide whether it should be installed, as something else needs it. But the leaf packages, they can decide directly whether they want them without affecting the operation of other software.

It should just print the package names, one per line.

`pc-missing`

The `pc-missing` command takes 2 arguments: the names of two package sets. For example:

```
$ pc-missing wanted current
```

This command takes 2 package sets, A and B, and prints all packages that are in set A but **not** in set B (the *missing* packages). This can be used to find the packages that are wanted but not yet installed (as above), or the packages that are installed but not listed as wanted (extra packages, found by swapping `wanted` and `current`).

In set theory notation, this script prints $A \backslash B$.

It should just print the package names, one per line.

`pc-missing-leaves`

This command is like `pc-missing`, but it does not consider all packages in set A. Instead, it considers only the *leaf packages* in set A that are not installed in set B.

This script must still consider all packages in set B: a leaf in set A might be installed in B, but also be a dependency of another package in B. In this case it is not missing, because it is still installed.

In set theory notation, this script prints $\text{leaves}(A) \backslash B$.

`pc-versions`

This command takes two package sets and prints a 3-column output: the name of each package, and its version in each package set. Only packages in both package sets need to be listed.

Sample Data and Tests

For your benefit, we have provided 4 sample data files and some basic tests.

The data files are as follows:

- `base.mf` contains a small set of packages
- `vim.mf` contains the packages in `base.mf` with the following changes:

  - the package `vim-enhanced` and its dependencies are added
  - the package `xz-libs` has a different version

- `www.mf` and `sonar.mf` are the full package manifests for a couple of servers I run.

The tests are in the `t/basic-test.t` file. This is a shell script; run it from the root directory of the assignment:

```
$ ./t/basic-test.t
```

If successful, it should say `ok` for all 8 tests.

We will be using tests like those in `basic-test.t` to help with grading. Make sure that your scripts pass `t/basic-test.t` so that we can run them in the test harness.

Submitting

In addition to the four scripts, write a file `README.txt` that contains any relevant information, such as the URLs of external resources that you used.

Submit your scripts as a single gzipped `tar` file to TRACS. You can make such a file with the following command:

```
$ tar -cvzf assignment1.tgz README.txt pc-*
```