Jinx

Henk P. Penning
Computer Science Department, Utrecht University

February 15, 1991

Contents

1 Introduction

Jinx is an interactive tool for manipulating simple databases.

A Jinx-database consists of an arbitrary number of records. A record consists of fields. A field may contain arbitrary ASCII printable data (no newlines or tabs). All records in a single database have the same number of fields. A database may therefore be viewed as a row of columns. Associated with each column is a unique name (the fieldname). There is no limit on the number of columns in a Jinx-database. In a picture:

	column 1	column 2	column 3		column n
	name 1	*name 2*	*name 3*		*name n*
record 1	field 1	field 2	field 3		field n
record 2	field 1	field 2	field 3	...	field n
record 3	field 1	field 2	field 3		field n
...	...	...	...	...	...
record $k-2$	field 1	field 2	field 3		field n
record $k-1$	field 1	field 2	field 3	...	field n
record k	field 1	field 2	field 3		field n

Associated with each column is a regular expression which is used as a constraint on the data that may be entered in the column. Users may override constraints when entering or modifying data.

With Jinx you can manipulate at most one database at a time. This database is called the *current* database. Jinx enables the user to

- define a new database by adding fields to an existing or empty database.
- open a database on disk to make that database the current one. The current database is a copy of the database on disk.
- look at the raw data in the current database and move around in it by specifying a search pattern, setting a record number or paging back and forth.
- inspect the current database at record-level. Records can be modified, added, copied and deleted. You can move around by searching for a specified pattern in the data etc.
- create a new database by selecting certain columns from the current database. This is called *projection*.
- create a new database by selecting records from the current database by specifying regular expressions for one or more fields. Any record with a field matching the given expression for that field is selected. This is called *extraction*.
- create a new database by joining the current database with another database.
- sort the current database.
- copy records in from another database.

- check the uniqueness of entire records or the combination of one or more fields.
- compute new values by specifying a Perl expression for one or more fields.
- write the current database to disk. This is called *saving* the database.

When the user creates a new database by projection, extraction etc, the new database becomes the current database.

Jinx marks the current database as *modified* or *not modified*. A database is only marked *not modified* if it is an exact copy of a database on disk. After opening a database or saving one, the current database is marked *not modified*. Newly created databases are all marked *modified* initially. When the current database is marked *modified* and the user requests an action (like quiting Jinx, joining, projecting) that will scratch the current database, Jinx will ask permission first.

When on disk, a Jinx-database consists of two files: *name*`.des` containing the descriptor and *name*`.dat` containing the data (see section 5 for a description of their contents). Jinx uses *name* to identify a database. When opening a database a name has to be specified. That name becomes the name associated with the current database. Newly created databases are nameless. Changing the descriptor (fieldnames and/or patterns) will make the current database nameless. The user must specify a name when a nameless database is to be saved. That name becomes the name associated with the current database.
You can use Jinx by typing:

	`jinx` [*name1 name2* ...]
or	`jinx -L` *logfile* [*name1 name2* ...]
or	`jinx -L`*level logfile* [*name1 name2* ...]
or	`jinx -D`
or	`jinx -v`

If a namelist is given, Jinx tries to open database *name1*. The other names are are presented as defaults when the opening of another database is requested. If no namelist is given, Jinx just starts up without a current database. With the L-option information is logged to the specified logfile. By default, Jinx will log activities like opening and saving of databases. Loglevel 1 or 2 are only there for debugging purposes and will log many activities related to the presentation.

The D-option will make Jinx dump core for undump-purposes. The v-option will show version and copyright information.

When in Jinx, you usually see the following things on the screen. At the top is information about the current database. At the bottom of the screen are a menu, the status line and a command line. The rest of the screen is used for presenting info in the database. The amount of info shown depends on the size of the screen.

2 Operations

Jinx is menu driven. Commands in menus are single characters (key strokes). If the menu shows the entry `ex)it`, it means you will have to type '`x`' when you want to *exit*. If the menu shows the entry `RET=foo`, you will have to type RETURN when you want to *foo*. If the menu shows the entry `TAB=p)arty`, typing TAB or '`p`' will make Jinx *party*.

Jinx always presents you with a default command. By typing RETURN you choose the default. Instead of the commands $\Leftarrow$, $\Downarrow$, $\Uparrow$, $\Rightarrow$, you can use the `hjkl`-commands. If arrow-keys don't work on your system, you should be able to use the control-characters `^B` (Back), `^N` (Next), `^P` (Previous), `^F` (Forward). In short:

$\Leftarrow$	$\Downarrow$	$\Uparrow$	$\Rightarrow$	in any mode
`^B`	`^N`	`^P`	`^F`	in any mode
`h`	`j`	`k`	`l`	not in edit-mode

Typing '`?`' to a command prompt, places you in help-mode. In help-mode, you can get one line of text explaining some command by typing in the command. You exit help-mode by typing RETURN or space. In general, by typing TAB you leave the current menu with the least damage done. Use it to back out of situations where you don't want to be.

When you need to enter or modify a string, Jinx places you in edit-mode. The cursor can be moved by $\Leftarrow$ and $\Rightarrow$. Typing in the 'kill' character (as defined by *stty*), will delete the 'character under the cursor'. Typing in the 'erase' character will delete the 'character before the cursor'. Other characters will be inserted except that RETURN, TAB, $\Uparrow$ and $\Downarrow$ will make you leave edit-mode. When you type 'erase' in the first position of the string, the string is swapped with an (initially empty) backup string. Sometimes when you are requested to enter some string, a default is presented. You can discard the default by typing 'erase' and then type in your own stuff. Finish by typing RETURN. Any string you edit can be as long as you want. When the length of the string exceeds the space reserved on the screen for editing the string, the string will scroll in a peculiar manner. Try it by opening a database with a loooong name.

2.1 The meta-menu

After startup, Jinx presents the meta-menu. It mainly contains commands which operate on a database as a whole like opening, joining, sorting etc. If there is a database current, the data part of the screen will show the data in the database on a record per line basis. The data is there to give a global overview. When you have 'long' records that don't fit on the screen, you can move the view on the data right (left) by typing $\Rightarrow$ ($\Leftarrow$). Data may also be layed out in columns (see *View toggle*, section 2.1.16).

You can page back- and forward through the data with the commands *prev* and *next*. The current record is indicated by a '`>`' in the first column. Move to the next (previous) record with $\Downarrow$ ($\Uparrow$).

2.1.1 goto

Records in the database are numbered from 1 to whatever it takes. The *goto* command asks for a record number and makes it the current record. You can use $+N$ and $-N$ to indicate a number relative to the number of the current record. Finish entering the number with RETURN. With $\Uparrow$ and $\Downarrow$ you can step through numbers entered previously. With TAB you can indicate that you don't want to *goto* after all.

2.1.2 search

The *search* command asks for a search pattern and matches records starting with the record following the current record. If no following record matches, preceeding records are tried starting with record 1. If a record matches the pattern, it is made the current record. Finish entering the pattern with RETURN. With $\Uparrow$ and $\Downarrow$ you can step through patterns entered previously. With TAB you can indicate that you don't want to *search* after all.

2.1.3 re-search

The *re-search* command repeats the last search command.

2.1.4 inspect

With *inspect* you enter the main-menu which is described in detail in section 2.2. In the main-menu the database is shown on record-by-record basis. In it are commands to update, copy, add, and delete records.

2.1.5 Descr

With *Descr* you enter the descriptor-menu which is described in section 2.4. In the descriptor-menu the descriptor of the current database is shown. In it are commands to modify fieldnames and constraint patterns.

2.1.6 Project

Project enables you to make a new database consisting of certain columns of the current database. It shows the names of the columns you can choose. You can *select* a column and *delete* it again later. The ranking of each selected column is shown. You can *increment* and *decrement* the ranking of a column. With *show* you can get the current rank list. Finish selection with *finish*. With TAB (or *exit*) you can exit *Project* altogether.

Deleted records, kept for retrieval, are projected too.

2.1.7 Join

Join enables you to make a new database by joining the current database with another one. It asks for another database to join with. With $\Uparrow$ and $\Downarrow$ you can step through names of databases you joined with earlier. By typing TAB you can exit *Join* altogether.

When joining databases a and b, Jinx looks at the columns with names that appear both in a and b. The join-key of a record consists of the concatenation of such fields.

When join detects a record in a with a join-key that doesn't appear in b it asks you whether you want to *delete* the record from the join or to *add* empty data for the record. You can also specify (with *Delete*) that such records must always be left out of the join, or specify (with *Add*) that empty data is to be added for all such records that may appear.

When join detects a record in a with a join-key that appears more than once in b it asks you what to do. You can specify that you wish to *add all* records of b in the join. You can also specify (*delete*) to leave that record out of the join. Alternatively, you can specify (*Add*) that such records in a

must be joined with all proper records in *b*, or specify (*Delete*) that all such records in *a* should be left out of the join.
You may also choose *show*. This option shows you the records in *b* that can be joined with. You can then *select* a subset in various ways. This is described in detail in the section on selection (2.3).

When join detects a record in the current database with zero or more than one records in *b* to join with you may also *exit* the entire join operation.

Deleted records don't take part in the join and are discarded.

2.1.8 Order

Order allows you to sort the database. It shows the names of the columns so you can choose them as sort keys. It is in fact the same selection process as is used in *Project* (section 2.1.6).

You can *select* a column and *delete* it again later. The ranking of each selected column is shown. You can *increment* and *decrement* the ranking of a column. With *show* you can get the current rank list. Finish selection with *finish*. Select TAB (or *exit*) if you don't want to *Order* after all.

When you don't select any fieldname as a sortkey, records will be sorted in lexicographical order.

The list of deleted records, kept for later retrieval, is not modified by sort.

2.1.9 Test all

Test all matches all fields of all records with their constraint patterns. Testing is started with the current record. When a mismatch occurs, the record is shown. The cursor points to the mismatching field and a message is given. You can choose to search for the *next* mismatch, search for the *Next* record with a mismatch, *update* the current record, or *exit* testing.

2.1.10 Add field

Add Field allows you to add a new field to the current database. You will be asked for a fieldname, a constraint pattern and a value for the new field. With $\Uparrow$ and $\Downarrow$ you can step through names, patterns and values entered earlier. Typing TAB will abort the operation. Fieldnames must be alphanumeric. All fieldnames must be unique.

The current database will be made nameless because the descriptor was modified. The database will be marked *modified*.

2.1.11 save

With *save* you can write the current database to disk. The current database must have a name *name*. The data is written to *name*`.dat`. The previous data file (if any) is moved to a *name*`.dat.save`. Because the descriptor is not modified, *name*`.des` is not written to. The format of these files is described in the section on implementation (sec. 5).

2.1.12 Save as

With *Save as* you can write the current database to disk. The current database may be nameless. You will be asked to provide a *name*. With ⇑ and ⇓ you can step through names entered earlier. With TAB you can abort the *Save as* operation. The descriptor will be written to *name*`.des`, and the data will be written to *name*`.dat`. The previous descriptor and/or data (if any) are moved to *name*`.des.save` and *name*`.dat.save`. The format of these files is described in the section on implementation (sec. 5).

If the current database is nameless, *Save as* will make *name* the name of the current database. If the current database has a name and is marked *modified*, *Save as* will not mark the current database as *not modified*.

2.1.13 open

With *open* you can read in a database from disk to make it the current database. You will be asked to provide a *name*. With ⇑ and ⇓ you can step through names entered earlier. With TAB you can abort the *open* operation.

2.1.14 Guess

With *Guess* you can build a Jinx database from data which already looks like a Jinx database. You will be asked for the name of a file and a pattern. With ⇑ and ⇓ you can step through names and patterns entered earlier. With TAB you can abort the *Guess* operation.

The specified file is opened for reading. Each line in the file is interpreted as a record and split into fields with the pattern. A descriptor with enough fields is generated. Fields are named `field1`, `field2`, etc. The constraint pattern for every field is '`.*`'.

You can make an example database with *Guess* using file `/etc/passwd` and pattern '`:`'.

2.1.15 Clear

With *Clear* you can make Jinx forget about the current database: no name, no data, no descriptor anymore.

2.1.16 View toggle

View toggle switches between 'raw mode' and 'column mode'. In 'raw mode' Jinx shows the raw data. In 'column mode' Jinx presents the data in the current database aligned in columns, with purely numeric data shown right-aligned, and other data shown left-aligned. In 'column mode', the command $\Rightarrow$ ($\Leftarrow$) will move the view on the data one column to the right (left).

2.1.17 quit

With *quit* you can leave Jinx. You will be warned if the current database is marked *modified*.

2.1.18 Quit

Use *Quit* to leave Jinx without further ado.

2.2 The main-menu

After *inspect* in the meta-menu, Jinx shows the main-menu. It presents commands which operate on the current database on a record-by-record basis. The data part of the screen will show (part of) the current record, one field per line. When you have 'long' fields that don't fit on the screen, you can move the view on the record to the right (left) with $\Rightarrow$ ($\Leftarrow$).

You can move to the next (previous) record with *prev* (*next*). The current field is indicated by a '`>`' in the first column. Move to the next (previous) field with $\Downarrow$ ($\Uparrow$).

2.2.1 goto

The *goto* command asks for a record number and makes it the current record. You can use $+N$ and $-N$ to indicate a number relative to the number of the current record. Finish entering the number with RETURN. With $\Uparrow$ and $\Downarrow$ you can step through numbers entered previously. With TAB you can indicate that you don't want to *goto* after all.

2.2.2 search

The *search* command asks for a search pattern and tries to find a match. First it looks at the fields following the current field. If this fails, it matches following records. If this fails too, it matches preceeding records starting with record 1. If a record matches the pattern, it is made the current record and the matching field is made the current field. Finish entering the pattern with RETURN. With $\Uparrow$ and $\Downarrow$ you can step through patterns entered previously. With TAB you can indicate that you don't want to *search* after all.

2.2.3 re-search

The *re-search* command repeats the last search command.

2.2.4 update

Choosing *update* lets you modify the current record. Move to a previous (next) field with $\Uparrow$ ($\Downarrow$ or RETURN). Finish editing the record with TAB. If you modify a field, the field will be checked with the constraint pattern for that field. If a mismatch occurs you are asked what to do. You can *edit* the field again, or you can tell Jinx to *forget* about the modification and leave the field as it was. You can also tell Jinx to *accept* the field as it is, ignoring the mismatch. You may also *exit* the entire update operation, leaving the record as it was.

After you finish editing the record with TAB you will be asked to confirm the update. Again, you may *edit* some more, *accept* the record as it is, or tell Jinx to *forget* about the update and leave the record as it was.

Only the modified fields are checked. If some 'improper' field value was accepted earlier, Jinx won't complain as long as that value isn't changed.

2.2.5 add

With *add* you can enter a new record. You are presented with an empty record that you can modify as with *update*. When you finish editing (with TAB), Jinx will ask you what to do if there are 'improper' empty values that you haven't explicitly accepted.

2.2.6 copy

With *copy* you can enter a new record. You are presented with a copy of the current record that you can modify as with *update*. When you finish editing (with TAB), Jinx will ask you what to do if there are 'improper' values that you haven't explicitly accepted.

2.2.7 delete

With *delete* you can delete the current record.

2.2.8 test

The *test* command matches all fields of the current record with their constraint patterns.

2.2.9 Test all

Test all works like the same command in the meta-menu (section 2.1.9). It matches all fields of all records with their constraint patterns. Testing is started with the current record. When a mismatch occurs, the record is shown. The cursor points to the mismatching field and a message is given. You can choose to search for the *next* mismatch, search for the *Next* record with a mismatch, *update* the current record, or *exit* testing.

2.2.10 Extract

Extract lets you mark and select a subset of the records in the database with the select-menu (see 2.3). All but the selected records are deleted. For safety, the current database is made nameless. The deleted records can be retrieved with *Undo* (see 2.2.12).

2.2.11 Delete

Delete lets you mark and select a subset of the records in the database with the select-menu (see 2.3). All the selected records are deleted. The deleted records can be retrieved with *Undo* (see 2.2.12).

2.2.12 Undo

Undo lets you mark and select a subset of the deleted records with the select-menu (see 2.3). All the selected records are put back in the database again. Jinx forgets about deleted records if you scratch the current database by opening another one, joining, etc.

2.2.13 Compute

Compute lets you compute new values in the database. You can enter a Perl expression for each field. For each field, the default expression shown is the last non-empty expression entered earlier in the session for that field. When doing the computation, empty expressions are ignored. An expression consisting only of white space is an error. To be precise, if expression *expr* is specified for field *fieldname*, the result of

do { *expr* ; }

is assigned to field *fieldname*, so *expr* may actually be a list of statements. Assignment to the current record is done after all the expressions are evaluated.

The expression is evaluated in a context where the following variables are defined:

`__RECNUM__`	record number of the current record
$*fieldname*	value of field *fieldname* in the current record

During the computation for one record, these variables keep their value unless you change them yourself. In the deleted records, new fieldvalues are computed too.

I wanted to include the following but the implementation caused Perl to crash.

> Before the computation of the first record, a `reset('[a-z]')` is done. This implies that variables starting with a lower-case letter are reset, while variables starting with an upper-case letter retain their value from one *Compute* to another.

It will be included in Jinx as soon as the patch comes out.

Compute can screw up Jinx and your data in many ways. Introducing TABs and/or newlines will ruin the presentation. Introducing newlines in fields will corrupt the record structure when you write the database to disk. Accessing variables in package *main* may crash Jinx.
It is hard to decide what action to take if something like this happens. One thing has been taken care of though. If a computed field value contains the value of *$;*, it is replaced by `$;` because *$;* is used in Jinx as a field separator in records.

There is no way (yet) to access fields in other records than the current record. Also it is not (yet) possible to bind an expression to a field from one session to another.

2.2.14 Peek

Peek lets you read records from another database. It asks for the name of a database *other*. After opening *other*, it lets you mark and select a subset of the records in *other* with the select-menu (see 2.3). In the select-menu, field-names in *other* that do not appear in the current database are preceeded by two underscores (__). These fields will not be copied to the current database because there is no column for them. Selected records of *other* are added at the end of the current database, but some sort of projection and reshufle of fields is done first. Only the fields with a name appearing in the current database are copied, others are left out. Empty data is added for fields in the current database that do not appear in *other*. This implies that you can only Peek at a database which has one or more fieldnames in common with the current database. From those databases, only the data in columns with common names can be copied.

2.2.15 Read

Read works like *Peek* except that it strips from *other* those records which are already in the current database.

2.2.16 Key test

Key test lets you run a uniqueness test on the current database. You are asked to select one or more fields in the current database (just as in *Order* and *Project*). *Key test* combines these fields for each record to produce a key for each record. Then it tests if every record has a different key. If some records have the same key, you enter a menu where you can specify (with *show to delete*) that you want to select some records for deletion. (See section 2.3 on how to make a selection.) Alternatively, you can ask for the *next* key which appears in more that one record, or *finish* the *Key test* operation deleting records as indicated sofar, or *exit* the operation without any change.

If you don't select any fields, *Key test* will report on records that appear two or more times in the current database.

2.2.17 finish

With *finish* (or TAB) you can leave the main-menu.

2.2.18 Quit

With *Quit* you can leave Jinx without further ado.

2.3 The select-menu

The select-menu lets you select records from a list of records. Depending on the context, selected records will be used by *Delete*, *Extract*, *Undo*, *Join*, etc.

In the select-menu, records are presented as in the main-menu. You can order the *next* or *previous* record, *search* with a pattern and *re-search*, or *goto* a record with a specific number.

A record is either *marked* or *unmarked*. Initially all records are unmarked. With *toggle* you can change the marking of the current record. *Toggle* toggles the marking of all records. *Clear* makes all records unmarked again. Use (*Clear*,*Toggle*) to mark all records.

With *Mark* you can specify a pattern (regular expression) for each field. Every record with a field matching the corresponding pattern is marked. Empty patterns are left out of the marking process. *Unmark* is like *Mark* except that marks are removed.

With '<' (*previous mark*) and '>' (*next mark*) you can step through the records marked sofar. With '(' (*previous unmark*) and ')' (*next unmark*) you can step through the unmarked records.

When you are done marking, you can either *select* the current record, *Select all* marked records, or *exit* to abort the selection process altogether.

2.4 The descriptor-menu

After *Descr* in the meta-menu, Jinx shows the descriptor-menu. It presents commands which allow you to change the attributes (fieldnames and constraint patterns) of the columns in the current database. The data part of the screen will show the fieldnames and constraint patterns of the descriptor. The current fieldname is indicated by a '>' in the first column. Move to the next (previous) fieldname with $\Downarrow$ ($\Uparrow$).

You can leave the descriptor-menu with *finish*. With TAB (or *exit*) you can abandon all modifications.

If you modify the descriptor, the current database will be made nameless because the descriptor on disk (if any) is not in accordance with the descriptor of the current database.

2.4.1 pat edit

With *pat edit* you enter edit-mode in which you can edit the constraint patterns like ordinary field-values (see section 2.2.4 on updating). The only difference is that you can only enter valid regular expressions. You can't make Jinx *accept* illegal regular expressions.

2.4.2 name edit

With *name edit* you enter edit-mode in which you can edit the fieldnames like ordinary field-values (see section 2.2.4 on updating). The only difference is that you can only enter unique alphanumeric fieldnames. You can't make Jinx *accept* illegal fieldnames.

3 Tools

See `man jinx` for the description of some support tools.

4 Examples

Start up Jinx by simply typing `jinx`. When you don't have a Jinx database ready, make one with *Guess*. Try file `/etc/passwd` with separator '`:`'. Write the database to disk with *Save as* as `blob`. Update the descriptor with *Descr*. Change the names into something meaningful. Now, make two new databases *hidden* and *rest* consisting of the (login,passwd) columns and of the (login,uid,gid,name,home,shell) columns. If you succeed, try to *Join* them again and save the result as `blib`. Diff `blob.dat` and `blib.dat`. They should be the same.

Now, suppose a computer science department has a secretary where students go when they come to study. Every student has a student-card with a unique number on it. The secretary keeps an address database mapping card-ids to info about the bearer like name, address, etc.
Suppose a teacher wants to keep a record of every student following some course. The teacher can open the address database and extract the relevant records, project out the irrelevant fields, add some new fields, and save it as the course database. If a student comes to enroll later, the teacher can Read in the address database again to add the student's record to the course database.

5 Implementation

A Jinx-database is represented on disk as follows. The file *name*`.des` contains lines of text like

attribute-name : *column-number* : *attribute-value*

where *attribute-name* can be `name` or `cpat`. The corresponding *attribute-values* represent fieldnames and constraint patterns for column *column-number*. Other attributes may be introduced later.

The file *name*`.dat` contains lines of text, one for each record.

In both files fields are separated by a colon (‘`:`’). The colons in user data and attribute values are escaped by exclamation marks (‘`!`’), as are the exclamation marks in user data and attribute values. (I know I could have used tabs as field separators and no escapes since tabs are forbidden in user data anyway. I didn’t because I hate tabs and wanted to keep the files readable :–).

Various things may change sooner or later. However, the format of the files is flexible enough to allow transparant upgrades.

6 Installation

Jinx requires the *cterm* package which provides an interface to *curses*. BSD-curses might not be good enough. Use ATT-curses if you have it. Cterm works fine on HP/UX, SunOs (if /usr/5bin/cc is used) and others. Ultrix has it’s problems but should work. Jinx also requires plain Perl (PL18 and up).

Install cterm if you don’t have it. See section 7 on how to get it. Unpack the Jinx stuff in a separate directory. If your Perl isn’t in `/local/bin/perl`, change the first line in `jinx`, `Jjoin`, `Jlist`, `Jproject`, `Jreport` and `Jsort` to the proper `#!`-line.

Start Jinx by typing `jinx`. This may take some time. Jinx checks cterm to see if all the necessary curses functions and constants are defined in the cterm interface. If some required function is missing (or has a different name), look at the installation procedure for cterm on how to add a function to cterm. Leave Jinx with ‘`q`’ if it starts up at all.

Study the Makefile. Edit the variables `PERLLIB` and `BINDIR`. They determine where the Perl libraries and programs go. Edit the variables `LIBMODE` and `BINMODE`. They determine the permissions of the installed Perl libraries and programs. Edit the names of the programs used in the Makefile if necessary.

If you have undump, make a binary version of Jinx with `make dump`. Install Jinx with `make dump-install` if you are successful. If you don't have undump, install Jinx with `make install`. Install the manual page `jinx.1` by hand. If you have LaTeX, do `make doc` and print `jinx.dvi`. If not, pick up a PostScript version of the tutorial `jinx.ps`. Read it and/or play.

7 How to get Jinx/cterm

Jinx, a (PostScript) jinx tutorial, and cterm are ftp'able from archive.cs.ruu.nl (`131.211.80.5`). Look in directory `pub/UNIX` for the files `jinx.shar.Z`, `jinx.ps.Z`, and `cterm.shar.Z`. If you don't have access to ftp, send a message to `mail-server@cs.ruu.nl` containing:

```
begin
send cterm.shar
send jinx.shar
send jinx.ps
end
```

The compressed, uuencoded stuff will be sent to you.

8 Copyright, Warranty