NAME

ppmkpat - make a pattern (pattern prepass, optimizer and compiler)

SYNOPSIS

ppmkpat [patname] [options] < patdefinition

DESCRIPTION

Ppmkpat creates (makes) common pattern package patterns from pattern definitions and previously defined (predefined) patterns.

The following is a brief description of the command line options:

+d The next argument is a pattern directory to be added to the directory search order.

+fo Create the pattern in object format.

+fs Create the pattern in standard format (this is the default if **+fo** is not present).

+ipok Output to stderr the characters **IP** when ppmkpat starts and the characters **OK** when ppmkpat finishes if the pattern was created with no errors.

+p Perform only the prepass on the pattern definition and output the results to stdout. This is the same as performing the cc(1) command with the -E option.

-p The prepass is not performed on the pattern definition.

+r Restrict the definition to a subset of the normally allowed built-in patterns. This is used by the ppsccsgp(3L) subroutine to prevent the use of special built-in patterns (e.g., frepeat and eofrpt). The pattern compiler source header files should be consulted for a list of the specific built-in patterns which are affected.

+t Perform a translation by mapping lower case string characters into upper case characters. This translation is only performed on string argument characters.

-s Do not include pattern source definition in the compiled output. This can be used to save space, and increase security.

+s Include pattern source definition in the compiled output (done by default).

-D Similar to the -D option of cc(1). May be used as -Dsymbol or -Dsymbol=value, where **value** is a number, or an unquoted

string. The #if - #endif, and #ifdef - #endif preprocessor directives will recognize these **-D** options. Multiple **-D** options are allowed.

Ppmkpat reads the the pattern definition <patdefinition> from standard input <stdin>. The predefined patterns are read from files. The directories which are searched for the predefined patterns are controlled by **+d** options in the command line. If no **+d** options are specified then the following directory search order is used:

/keypat	**builtin pattern/keyword/primitives directory**
.	**present working directory**
/compat	**common pattern directory**
/usr/pat	**common user pattern directory**

If one or more **+d** options are present in the command line, then all of the default directories will be removed from the search order except **/keypat**. The keyword directory (/keypat) may never be removed from the search order.

The argument directly following a **+d** (**+d** <dirname>) is the path name of a directory. This path name is added to the end of the search order list.

The following example should explain the [redacted] **+d** option discussed above:

ppmkpat +d . +d /type01/pat +d /compat +d /usr/pat

For the command line above the directory search order will be:

/keypat
. **(present working directory)**
/type01/pat
/compat
/usr/pat

Ppmkpat creates a pattern with one of several formats.

Standard format - This form will be produced by default (i.e. no **+fo** in the command line).

Object format - This form will be produced if the **+fo** option is used in the command line.

Ppmkpat puts its compiled output (the pattern) into a file. If **patname** is specified in the command line, then some characters are appended to the end of patname and used as the name of the

pattern file. If **patname** is not specified, then a default name (PPDFLTNAM as defined in the **/usr/include/ppsubs.h** header file) is appended with two characters and used as the name of the file. The characters which are appended to the file name are **.p** for standard format and **.o** for object format.

The object formated file may be link loaded into an **a.out** file like any other object file.

FILES

/keypat	builtin pattern/keyword/primitives directory
/compat	common pattern directory
/usr/pat	common user pattern directory
temp.p	default standard output pattern file
temp.o	default object format (both types) pattern file
/tmp/ppsrc<prid>.c	temporary file for prepass; <prid> = process ID

SEE ALSO

a.out(5), pattern(5L) ppdpat(1L)

DIAGNOSTICS

The diagnostics produced by **ppmkpat** are intended to be self explanatory. Ppmkpat exits with value PPSYNTAXERR if one or more errors were found in the definition.

BUGS

Ppmkpat will accept a long (over 60 characters in length) **patname**. However the object formats use the first seven characters after the last / character in **patname** as the **C** symbol name of the pattern (e.g. if **patname** = **/compat/sctab012**, then **sctab01** will be the symbol name of the pattern when it is part of an **a.out** file).

FILES

/keypat	builtin pattern/keyword/primitives directory
/compat	common pattern directory
/usr/pat	common user pattern directory
temp.p	default standard output pattern file
temp.o	default object format (both types) pattern file
/tmp/ppsrc<prid>.c	temporary file for prepass; <prid> = process ID

SEE ALSO

a.out(5), pattern(5L) ppdpat(1L)

DIAGNOSTICS

The diagnostics produced by **ppmkpat** are intended to be self explanatory. Ppmkpat exits with value PPSYNTAXERR if one or more errors were found in the definition.

BUGS

Ppmkpat will accept a long (over 60 characters in length) **patname**. However the object formats use the first seven characters after the last / character in **patname** as the C symbol name of the pattern (e.g. if **patname** = **/compat/sctab012**, then **sctab01** will be the symbol name of the pattern when it is part of an **a.out** file).

NAME

ppvpat - pattern package reverse compiler

SYNOPSIS

ppvpat <patfilename>

DESCRIPTION

ppvpat outputs the pattern definition by reverse compiling the part of a common pattern package pattern file.

The argument <patfilename> is the name of the pattern file, including the format dependent ending (ex. temp.p).

FILES

/usr/include/ppsubs.h common pattern package header file
pp_kname.h header file inialization pattern keywords array
pp_kval.h pattern primitives value header file

SEE ALSO

ppmkpat(1L), pattern(5L)

DIAGNOSTICS

The diagnostics produced are intended to be self-explanatory.

NAME

progress - progress report handler

SYNOPSIS

progress

DESCRIPTION

progress is an iterative command that provides the following functions across the networked SCCS systems:

1. Create a report. This is used to create an individual progress report. Users are given the option to save previous reports. The entire report is distributed across all systems.

2. Edit an existing individual report. This is used to correct or add to an already existing individual progress report. The entire report is distributed across all systems.

3. Add information to a group summary report. Only the new information is distributed across the systems where it is then appended to the full report.

4. Modify an existing group report. This must be done with caution as the entire group report is redistributed. A race condition can develope with updates to the same report from different machines.

5. View a report. The output of an individual or group report can be sent to a file, a line printer, or to your terminal.

6. Update a distribution list. This includes distribution lists for all group summary reports and individual progress reports from any given supervisory group. Only the new information is distributed across the systems where it is then appended to the distribution list.

NOTES

The command will prompt the user for any needed information such as individuals initials, supervisor's initials, etc. The command will continue prompting for the next function to be performed until the user requests the command to stop.

SEE ALSO

progressd(1L)

FILES

/progress/dist

/progress/<supervisor initials>/dist

/progress/<supervisor initials>/group<montn>

/progress/<supervisor
initials>/old_summarys/group<montn><year>

/progress/<supervisor initials>/<individuals initials>

NAME

putsrc - put source code on proper update pack.

SYNOPSIS

putsrc [+o] <file_name> <pr_directory> <specific_make_name>

DESCRIPTION

This routine puts source code on the appropriate update pack. This is accomplished by using the <specific_make_name> argument to access a special data file containing control information. This file contains the update packs to be utilized for each specific make. A search is made of all remaining update packs to determine if this file already exists. The user is informed if the file is found.

EXAMPLE: putsrc +o gen_rng01.c pr-ip135 SC5I6

NOTE: The user must ensure that all valid update and source packs are mounted in order for this routine to run properly. In addition the update pack must be writeable from the computer being used.

OPTION	EFFECT
+o	If <file_name> exists in the target directory, overwrite it with new file and inform user. If this option is not set the routine will ask the user if the existing file is to be over written.

Entering the command name (putsrc) alone, will result in a printout of the command format and a listing of all accepable make names.

FILES

/usr/bin/putsrc /usr/bin/sc11 /usr/bin/sc12

SEE ALSO

getsrc(3L)

BUGS

NAME

net − execute a command on a remote machine

SYNOPSIS

net [**−m** machine] [**−l** login] [**−p** password] [**−r** respfile] [**−**] [**−f**] [**−n**] [**−q**] command

DESCRIPTION

The *net* command sends the specified *command* (which should be enclosed in quotes) over the network to the specified (or default) remote machine. The network will notify the user when the command has been executed and will return to him any output or error indication by 'writing' (see *write*(1)) to the terminal if he is still logged in, or 'mailing' (see *mail*(1)) otherwise.

There are a number of options, which must precede the command. Options may be specified on the command line, preceding the command, or in a file ".netrc" in the user's login directory. The ".netrc" file is not described here. The **−m** option specifies the desired remote machine. If a remote machine is not specified, the default one is used. The machine name may be a one letter abbreviation or a full name; upper− and lower−case distinctions are ignored. If the standard output and standard error files are to be saved, the **−r** option returns to the originating user a file *(respfile)* containing the standard output and error files when the command was executed on the remote machine. If this option is used, no message is written back. The presence of a non-zero length *respfile* indicates completion. The **−q** option suppresses all acknowledgements unless an error occurs, there is output from the command, or the exit code of *command* is non-zero.

If the **−l** and **−p** options are not specified, and the login name and password are not in the ".netrc" file, a remote login name and password is prompted for on the terminal; the **−f** flag forces login name and password prompting. A single **−** indicates that the standard input from the local machine is to be taken and transmitted to the remote machine, where it will be the standard input for *command*. The **−n** flag forces all acknowledgment and output messages to be mailed rather than written on the terminal. Options do not need to be separated by spaces, i.e. either "−m C" or "−mC" is accepted. There are also other options intended to be used by higher level application programs and shell scripts only; they will not be described here.

The net command prepares a file to be sent to the remote machine and queues it in the 'network queue.' *Netq* (1) gives information about the queues.

AUTHOR

Eric Schmidt

FILES

/usr/spool/berknet/logfile	logfile with information about net activity
/usr/spool/berknet/plogfile?	log file including packet transmission statistics
/usr/spool/berknet/netstat?	statistics file
/usr/net/network.map	local network names and topology

BUGS

−q should be the default.

SEE ALSO

netrm(1), netq(1), netlog(1), netcp(1), netlpr(1), netmail(1), netlogin(1), mail(1)
"An Introduction to the Berkeley Network", by Eric Schmidt

NAME

netlpr − use a remote lineprinter through the net

SYNOPSIS

netlpr [**−m** machine] [**−l** login] [**−p** password] [**−f**] [**−q**] [**−n**] [**−c** command] [name1 ... namen]

DESCRIPTION

Netlpr sends the named files, (or the standard input if none are named), to a remote lineprinter; the **−m** option forces the files to be printed on the specified machine. (If not specified, the default machine is used.) The **−l**, **−p**, **−f**, **−q**, and **−n** options behave exactly as in *net*(1). If the **−c** option is specified, the *command* is used in place of 'lpr'. This allows the use of different lineprinters on the remote machine. See the file *'/usr/net/network.map'* for a list of available commands. Any other options are passed through to *lpr*(1) on the remote machine. Copies of the files are not made on the remote machine.

Netlpr executes the *net*(1) command.

FILES

/usr/net/network.map	lists the allowed local printer names

SEE ALSO

net(1), netrm(1), netq(1), netlog(1), netcp(1), netmail(1), netlogin(1), mail(1), lpr(1)

AUTHOR

Eric Schmidt

NAME

netq − print contents of network queue

SYNOPSIS

netq [**−a**] [machine]

DESCRIPTION

Netq lists the contents of the network queue, one request per line, for each directly-connected machine. For each request, it shows the login name and machine of the originator, the destination machine and login name, and the length (in bytes) of the request (this will be larger than any files transferred (e.g. by *netcp)*, because of header information). Also described are the queue filename which may be used as an argument to *netrm*(1), the time entered the queue, and the command being sent.

Netq summarizes requests by other users. If the **−a** option is specified, requests from all users are listed.

If a *machine* is specified, only the queue for that directly-connected machine is listed.

The requests are listed in the order they will be sent; the queue for each machine is totally independent from the other machine's queues.

AUTHOR

Eric Schmidt

FILES

/usr/spool/berknet/send?	the directories where the queues are
/usr/spool/berknet/logfile	the log

SEE ALSO

net(1), netrm(1), netlog(1), netcp(1), netlpr(1), netmail(1), netlogin(1), mail(1)

BUGS

Netq should also list files in net queues on intermediate machines.
The commands are sent shortest-job first. There is no way to delay a shorter, earlier request.

NAME

netcp − remote copy of files through the net

SYNOPSIS

netcp [**−l** login] [**−p** password] [**−f**] [**−n**] [**−q**] fromfile tofile

DESCRIPTION

Netcp copies files between machines and is similar to *cp*(1). At least one of *fromfile* and *tofile* must be remote. The **−l**, **−p**, **−f**, **−q**, and **−n** behave exactly as in *net*(1).

Fromfile and *tofile* follow these conventions:

1. A simple filename is assumed to be local and from the current directory.
2. A filename preceded by a machine designator (see below) is a reference to a file on the specified remote machine. If a full pathname is not given, it is assumed to be from the login directory.

Examples:

grades.p	file in the current directory on local machine
C:junk	file in your login directory on C
/usr/lib/pq	file on local machine
C:comp/c2.c	file in a subdirectory on C machine

When files are being "fetched", that is, the *fromfile* is remote and the *tofile* is local, the *tofile* is created zero-length mode 600. For security reasons, when the "fetched" file's contents arrive at the local machine, the file must still be zero-length and mode 0600. No confirmation is sent to the user that the file has been "fetched"; a non-zero file length indicates completion.

Netcp executes the *net*(1) command.

SEE ALSO

net(1), netrm(1), netq(1), netlog(1), netlpr(1), netmail(1), netlogin(1), cp(1), mail(1)

AUTHOR

Eric Schmidt

BUGS

The second filename may not be defaulted to a directory name as in *cp*(1), it must be given explicitly.
The file mode may or may not be set correctly.

NAME

netrm − remove a command from the network queue

SYNOPSIS

netrm [−] [name1 ... namen]

DESCRIPTION

Netrm removes files from the network queue which have been queued for transmission to remote machines (but not yet sent). The *names* specified are the filenames reported by the *netq*(1) command. The − option indicates that all files owned by the person logged in are to be removed.

Only the owner of the file or super-user can *netrm* the file.

AUTHOR

Eric Schmidt

FILES

/usr/spool/berknet/send? the directories where the queues are

BUGS

Files on network queues on intermediate machines cannot be removed.
There should be a **−m** flag to use with − to remove all your requests to one particular machine.

SEE ALSO

net(1), netq(1), netcp(1), netlpr(1), netmail(1), netlogin(1), mail(1)

NAME

progress - progress report handler

SYNOPSIS

progress

DESCRIPTION

Progress is an iterative type command that provides the following functions:

1. Create a report. This can be used to create both group summary reports and individual progress reports. When a group's summary report is created, a news item will be produced. For group summary reports, the previous month's reports are automatically saved. For individual progress reports, users are given the option to save previous reports.

2. Edit an existing report. This can be used to correct and append to an already existing group summary report or an individual progress report.

3. View a report. This can be accomplished by specifying the output to go to a file, line printer, or to your terminal.

4. Add a name to a distribution list. This includes distribution lists for all group summary reports and individual progress reports from any given supervisory group.

NOTES

The command will prompt the user for any needed information such as login initials, supervisor's initials, etc. The command will continue prompting for the next function to be performed until the user requests the command to stop.

SEE ALSO

progressd(1L)

FILES

/progress/dist

/progress/<supervisor initials>/dist

/progress/<supervisor initials>/group<month>

/progress/<supervisor initials>/old_summarys/group<month>

/progress/<supervisor initials>/<login initials>

NAME

progressd - daily progress report generator

SYNOPSIS

progressd

DESCRIPTION

Progressd prompts the user for any progress made since the last time the command was run. Any progress entered is appended to the end of the same file used in the "progress" command.

When the user wants to start a new month's progress report, the first day must be entered by using the "progress" command and then choosing the option needed to create a new file. All the rest of the days of the month can be done by "progressd".

SEE ALSO

progress(1L)

FILES

/progress/<supervisor initials>/<login initials>

NAME

pretty - adjust C code to coding standard

SYNOPSIS

pretty [-a|b|c[n:n:n ...]] [-rin[+]] [file ...]

DESCRIPTION

Pretty takes C code, either from the standard input or from the listed files and reformats them to the coding standard specified by the **-a**, **-b**, or **-c** switch. If no coding standard is specified, **-a4** is assumed.

-r Output is normally directed to the standard output. The **-r** switch directs pretty to place the output temporarily in tmppretty????? and at the end of reformatting, rename original to old.original and tmppretty????? to original, thus preserving a copy of the original code.

-i Normally pretty makes an attempt to add four extra spaces to any lines that it thinks are continuations of previous lines. If a previous line did not end in ; : { or } and the new line does start with a {, then pretty thinks it is a continuation. The **-i** switch tells pretty not to worry about continuation lines and no extra spaces are added to such lines.

-n[+]

Pretty can handle the C formatting macros (Programmers Note #113) if the **-n** switch is specified. This switch tells pretty that comments can start with _C*, _B*, or _F* as well as the usual /* sequence. It also tells pretty that braces might be found as _{ and _} and that the occurance of __ should be treated as a complete statement. The "+" additionally specifies that all indents must be multiples of 8 and that when deindenting switch statement labels, they should be deindented by 8 instead of the normal 2. This is necessary for anyone using the _{ and _} in place of normal braces and the __ to specify that nroff deindent the switch statment labels. This is because the nroff formatting macros do not delete leading spaces in lines, though they do delete leading tabs. Without the "+" option, the final results after nroff'ing would not be what was desired. It is suggested that people use the commenting aids of the nroff macros, but allow pretty to handle the braces and indenting. If this is done, only the **n** option without the "+" is required. It should be noted that the "+" on the **n** switch overides any tab setting given with the **a** , **b** , or **c** switches.

The coding standards are:

```
-a   xxxxxxxxxxxxxxxxxxxxxxx
     {
         xxxxxxxxxxxxxxxxxx
         xxxxxxxxxxxxxxxxxx
     }

-b   xxxxxxxxxxxxxxxxxxxxxxx{
         xxxxxxxxxxxxxxxxxx
         xxxxxxxxxxxxxxxxxx
     }

-c   xxxxxxxxxxxxxxxxxxxxxxx
       {
         xxxxxxxxxxxxxxxxxx
         xxxxxxxxxxxxxxxxxx
       }
```

A coding standard switch may be followed by an optional **n:n:....**, where each **n** is the size of the next indent in the series. The default has all indents set to 4 for coding standards **a** and **b** and 2 for coding standard **c**. This means that each new section of code will begin four spaces further to the right than the previous section. In the following example

pretty -c2:6:2 file

produces code of type **c** with the first indent at 2, the second at 2+6, the third at 2+6+2, and all later indents at 2 further in from the previous indent. Sample code would appear as

```
subroutine()
  {
        xxxxxxxxxxxxxxx
          {
            xxxxxxxxxxxxxx
            xxxxxxxxxxxxxx
          }
        xxxxxxxxxxxxxxxxx
        xxxxxxxxxxxxxxxxx
  }
```

Pretty makes certain assumptions that the user should be aware of. Lines beginning with # and comments beginning in the leftmost column are not adjusted. All other comments are indented normally. If the user translates a file of C code from types **-a** or **-c** to **-b** it is possible to move a { from inside an #ifdef , #if or #else to the first statement preceding that conditional. Code must not appear in this form or the following can happen:

-a code

```
        if (a > b)
#ifdef TYPE
        {
          xxxxxxxxxxx ;
          xxxxxxxxxxx ;
        }
#else
            xxxxxxxxxxx ;
#endif
```

-->

-b code

```
        if (a > b){
#ifdef TYPE
            xxxxxxxxxxx ;
            xxxxxxxxxxx ;
        }
#else
            xxxxxxxxxxx ;
#endif
```

As long as the code inside condtionals is syntactically complete, this problem will not arise.

If code is translated to first one coding standard and then back, there is the possibility of minor differences between the original and final output, caused by movement of blank lines. No functional changes in code appearance will take place.

Labels are checked for as the first printing item on lines. If they are encountered, the label is printed left justified on a line by itself to make it stand out. The keywords case and default of the switch statement are printed with the indent at that point reduced by 2 spaces if possible. Lines that do not appear to be complete, that is not ending in a ; : { or } character, cause the next line of code to be indented an extra 4 spaces as long as the new line doesn't begin with a { character. This feature is turned off by the **-i** switch.

FILES

tmppretty?????

NAME

read - read a character string from controlling terminal

SYNOPSIS

read string-variable

DESCRIPTION

Read reads a character string from the users terminal into one of ten string variables (a thru j) by opening and reading from `/dev/ln'. The strings then become available as arguments for subsequent commands which may reference them as $a thru $j.

Because the strings are available to any subsequent command executed by the shell the strings must be stored in temporary variables within the Shell. Read is therefore a command which is recognized and executed by the shell.

SEE ALSO

sh:o(1)

BUGS

NAME

recover - Recover files from Incremental Dump

USAGE

recover <filename> [<filename>...<filename>]

DESCRIPTION

recover allows the retrieval of lost/scrambled files from the Incremental archives (dumps) created daily @ 1:00am (the dump begins at 1:00; it may take several hours to complete, thus a file may not actually be saved until 3:00 or 4:00am). Incremental archives contain all files which were modified/created since the EPOCH Dump. They MIGHT NOT contain files which were removed (see below).

Each machine, DEV1 and DEV2 (scd1 & scd2), has 2 dedicated disk areas (numbered 1 & 2) utilized for Incremental Dumps. Each area is used on alternate days (the EPOCH Dump day is typically skipped) and thus provide a 2 day history of file system changes (ie. every other day, a given area is overwritten with new data). Their contents are referred to as the DAY1 and DAY2 Incrementals. See below for details of the files contained on an Incremental Dump.

The <filename> argument to **recover** should be the entire relative path name of the file you wish to recover. That is, the leading (root) '/' must be omitted (files in the incremental archive are referenced exclusively by their path name relative to the root dir '/'). If you are not sure where a file was, enter as much of the name as possible (since **recover** 'greps' a listing of Incremental contents for <filename>, this will limit the number of extraneous matches you must weed through).

recover will then:

Output a 'ls -l' of the latest 2 Incremental Contents Listings (list1 & list2 - corresponding to DAY1 & DAY2). Each listing is created at the time of the Incremental. Its modification date is the completion date/time for the last **successful** Dump for that day.

'Grep' the listings for the string (<filename>) provided. A match may be found in either, both or neither of the listings, for the reasons outlined below. If found, the mode/uid/gid/size of the file in the archive is output. This info and/or the date of the listing may be used to determine from which 'DAY' the file should be extracted.

Prompt the user for the necessary retrieval information:

Listing number where desired version found (1 or 2 - defaults to 1)

File name on Incremental (relative path name - defaults to <filename> arg)

New name of retrieved file (defaults to last component of Incremental file name, above)

Path of target directory for retrieved file (defaults to .)

Background? (defaults to no - you must wait for file to be retrieved)

The prompts are defaulted by entering a <Carriage Return> alone as a response.

recover will then retrieve the file from the Incremental archive corresponding to the list number entered (/etc/updfs is used).

[Directories may not be retrieved with recover. Consult a gnome when a large number of files must be recovered.]

NOTE:

Incremental dumps are only a 2 day history of file modifications. A given change made on a Thursday may be recovered before 1:00am Sunday AND, if no more mods are made, will be available through the following Friday. If further modifications are made, however, (eg. on Monday), the previous change will be lost after 1:00am of the 2nd day following the new change (ie. Wednesday). Confusing? - you bet.

If a file has not been modified/created since the EPOCH, it will not appear on the Incremental. Thus, files which have 'vanished mysteriously' cannot be recovered with recover unless they were changed/created since the EPOCH and had been removed (or vanished mysteriously) WITHIN the LAST 2 DAYS. Two days after a changed/created file is deleted, the Incremental image of the file systems will no longer contain the file.

The EPOCH packs must be consulted for retrieval of unmodified files and those modified files removed more than 2 days prior.

FILES

/incrdir/list1 /incrdir/list2 /dev/rDAY1 /dev/rDAY2

DIAGNOSTICS

'Cannot mount /dev/incrdir'. This is where the list? files reside. There are probably too many file systems mounted already.

BUGS

Note that Incrementals are not guaranteed to be successful every day. Perturbances in the file systems can cause the incremental areas of disk (56k blocks each - 2 per/machine) to be filled to capacity and not contain all the changes since the EPOCH. At-

tempts are made to ensure that the user directories are dumped first, but even this isn't guaranteed (perturbances include things like 'touch'ing every file in your directories - eg. by changing their modes/groups, etc. -or- copying an entire PR directory into your own).

Since the incremental directory, /incrdir, is a mounted file system, a failure to mount /dev/incrdir will result in an aborted Incremental Dump.

NAME

retrieve -- used to retrieve scripts, test files from save areas

SYNOPSIS

retrieve

DESCRIPTION

This program is used to retrieve test files from various storage file systems on the Load Test Machine. It is totally interactive with the user and it provides explanations of prompts when given a ?<cr> to a prompt. The user should know the name of the test plan associated with the test files they want if the files are stored in a restricted area.

SEE ALSO

save

DIAGNOSTICS

None

BUGS

Only one but, it is still hidden at this time.

NAME

rpg - run an rpg program

SYNOPSIS

rpg [options] <filename> [arguments]

DESCRIPTION

The "rpg" command is used to execute RPG programs from the UNIX shell. On some systems the command name may be "nrpg". Several options are available with this command:

+d - invoke dynamic debugger

-d - do not invoke dynamic debugger (default)

+t - turn on execution tracing

-t - do not turn on execution tracing (default)

+p - trace all "*switch", "*sccs", and "*xeq" commands.

-p - do not trace paths (default)

+w - print all warning messages (default)

-w - do not print warning messages

BUGS

RPG was a bug.

NAME

save -- save scripts, message, and other test files

SYNOPSIS

save

DESCRIPTION

This program will save scripts, message, and other types of files in either the restricted file system used by system test or in a non-restricted file system for developer files.

If the user wants to store files in a restricted file system, they must know the password to the save program. Normally, only system test coordinators will know this password. Storing files in a non-restricted file system does not require a password.

The program will prompt the user for any necessary input. If the user has a question concerning the prompts, just type a question mark followed by a carrage return and the program will give an explanation.

SEE ALSO

retrieve

DIAGNOSTICS

None

LIBRARY

None

BUGS

There is one but, it is unknown at this time.

NAME

sccmk - make/lint a No. 2 SCCS pident

SYNOPSIS

sccmk [**-flags**] [**+flags**] pgfile [source-dir]
sccmk [**-flags**] [**+flags**] gufile
sccmk [**-flags**] [**+flags**] aufile [aufile] ...

DESCRIPTION

The **sccmk** program will make/lint an au file by performing commands listed in the #SETUP, #LINT, #MAKE, #CLEANUP, and #COPY sections of an au file. It will make/lint au files listed in a gu file and process gu files listed in a pg file.

The flags to the **sccmk** program are turned on by preceding it with a plus (+) sign and turned off by preceding it with a minus (-) sign. The flags and defaults settings are as follows:

FLAG	DEFAULT	MEANING
c	on	Perform #COPY section of au file
m	on	Perform #MAKE section of an au file
K	off	Perform #LINT section of an au file, insert the -K option following all **lint** commands. The **c** and **m** flags are turned off.
S	off	Perform #LINT section of an au file, insert the -S option following all **lint** commands. The **c** and **m** flags are turned off.
V	off	Perform #LINT section of an au file. The **c** and **m** flags are turned off.
o	off	Compare originating and target files of the **move/cpmv** commands in #COPY section of an au file.
r	off	Do a replace in lieu of the **move/cpmv** commands in the #COPY section of an au file.
s	off	Do a restore in lieu of the **move/cpmv** commands in the #COPY section of an au file.
f	on	Perform #FINAL PIDENTS section of gu file
i	on	Perform #INITIAL PIDENTS section of gu file
p	on	Perform #PIDENTS section of the gu file.
x	off	Perform the #PREMAKE section of a pg file.
y	off	Perform the #MAKE section of a pg file
z	off	Perform the #FINAL section of a pg file
l	off	Perform statistical gathering tasks (not yet avail.)

The **sccmk** program will always execute the commands found in the #SETUP and #CLEANUP sections of the au file regardless of the **sccmk** options given.

Note in the above that when a **lint** option is given, the **c** and **m** flags are turned off. They can be turned back on by following the **lint** option with the desired **c** and/or **m** options, (e.g., **+Kmc**).

The **sccmk** program will examine the file argument(s) to determine

whether it is a pg, gu, or au file. A pg file must begin with a #PG section identifier. A gu file must begin with a #PR section identifier. An au file must begin with a #IDENTIFICATION section identifier. The **sccmk** program will chdir to the specified source-dir (**/src** if none is provided) before processing the entries within the pg file. While processing a gu or au file the **sccmk** program assumes all referenced files are in the current directory. **Sccmk** assumes the following:

- a maximum of 200 characters per line in a pg, gu, and au file. No checking is done, however.
- The #COPY section of an au file contains only the commands **move** and **cpmv**. This rule is enforced when performing a replace **(+r)** or a compare **(+o)** or a restore **(+s)**.

If either the **+r** flag or the **+s** flag is specified, the **+c** flag is ignored although the **sccmk** program still recognizes the **+c** flag as being on. If the **+r, +s** and **+o** flags are specified in any combination together, the restore will be performed first followed by the comparison and then the replace.

The following example will make the #MAKE section of every au file listed in the #PIDENTS section of the gu file given as an argument. The #COPY section of every "made" au file will do a **replace** instead of a **move** or **cpmv**.

```
sccmk -fi +r pr-1p137-01.gu
```

FILES

/bin/sh replace

SEE ALSO

autile(5), gufile(5), pgfile(5), sh(1), replace(?)

DIAGNOSTICS

If the **sccmk** command line options direct the command to execute command lines in the LINT and/or MAKE sections of a .au and these sections can not be found, then a diagnostic message will be output.

BUGS

A line of > 200 characters encountered while processing a pg, gu or au file can cause strange results, e.g., a core being dropped. If the **+r, +s,** or **+o** options are specified and a command in the #COPY section of an au file is continued across several lines, the maximum number characters assumed for that command is 250. If this limit is exceeded, strange things may happen, e.g., a dropped core.

All lines containing only blanks/tabs are thrown away by the **sccmk** program. Hence, in the #MAKE or #COPY section of an au file, a blank line should not be used after a continued line (one ending in a backslash).

NAME

sccmk - make a No. 2 SCCS program

SYNOPSIS

sccmk [**-flags**] [**+flags**] pgfile [source-dir]
sccmk [**-flags**] [**+flags**] gufile
sccmk [**-flags**] [**+flags**] aufile [aufile] ...

DESCRIPTION

The sccmk program will make an au file by performing commands listed in the #MAKE and #COPY sections of an au file. It will make au files listed in a gu file and process gu files listed in a pg file.

The flags to the sccmk program are turned on by preceding it with a plus (+) sign and turned off by preceding it with a minus (-) sign The flags and defaults settings are as follows:

FLAG	DEFAULT	MEANING
c	on	Perform #COPY section of au file
m	on	Perform #MAKE section of an au file
o	off	Compare originating and target files of the move/cpmv commands in #COPY section of an au file.
r	off	Do a replace in lieu of the move/cpmv commands in the #COPY section of an au file.
s	off	Do a restore in lieu of the move/cpmv commands in the #COPY section of an au file.
f	on	Perform #FINAL PIDENTS section of gu file
i	on	Perform #INITIAL PIDENTS section of gu file
p	on	Perform #PIDENTS section of the gu file.
x	off	Perform the #PREMAKE section of a pg file.
y	off	Perform the #MAKE section of a pg file
z	off	Perform the #FINAL section of a pg file
l	off	Perform statistical gathering tasks (not yet avail.)

The sccmk program will examine the file argument(s) to determine whether it is a pg, gu, or au file. A pg file must begin with a #PG section identifier. A gu file must begin with a #PR section identifier. An au file must begin with a #IDENTIFICATION section identifier. The sccmk program will chdir to the specified source-dir (/src if none is provided) before processing the entries within the pg file. While processing a gu or au file the sccmk program assumes all referenced files are in the current directory. Sccmk assumes the following:

- a maximum of 200 characters per line in a pg, gu, and au file. No checking is done, however.
- The #COPY section of an au file contains only the commands move and cpmv. This rule is enforced when performing a replace (**+r**) or a compare (**+o**) or a restore (**+s**).

If either the **+r** flag or the **+s** flag is specified, the **+c** flag is ignored although the sccmk program still recognizes the **+c** flag

as being on. If the **+r**, **+s** and **+o** flags are specified in any combination together, the restore will be performed first followed by the comparison and then the replace.

The following example will make the #MAKE section of every au file listed in the #PIDENTS section of the gu file given as an argument. The #COPY section of every "made" au file will do a replace instead of a move or cpmv.

```
sccmk -fi +r pr-1pi37-01.gu
```

FILES

/bin/sh replace

SEE ALSO

aufile(5), gufile(5), pgfile(5), sh(1), replace(?)

DIAGNOSTICS

BUGS

A line of > 200 characters encountered while processing a pg, gu or au file can cause strange results, e.g., a core being dropped. If the **+r**, **+s**, or **+o** options are specified and a command in the #COPY section of an au file is continued across several lines, the maximum number characters assumed for that command is 250. If this limit is exceeded, strange things may happen, e.g., a dropped core.

All lines containing only blanks/tabs are thrown away by the sccmk program. Hence, in the #MAKE or #COPY section of an au file, a blank line should not be used after a continued line (one ending in a backslash).

NAME

secprt - print section of prologue or PG files

SYNOPSIS

secprt [section ...] file ...

DESCRIPTION

The command secprt will print out the contents of a given section or sections of the listed files. The files given as arguments must be either prologue files or PG files.

The acceptable section arguments are:

COPY
DATA
IDENT
MAKE
PIDENT
PRDOC
PRINT
PROGRAM

Output is to the standard output, file descriptor two.

FILES

SEE ALSO

PG(5), au(5)

DIAGNOSTICS

BUGS

NAME

sed - stream editor

SYNOPSIS

sed [**-g**] [**-n**] [**-f** commandfile] ... [[**-e**] command] ... [file] ...

DESCRIPTION

Sed copies the input files (standard input default) to the standard output, perhaps performing one or more editor commands (see ed (I)) on each line.

The following flags are interpreted by sed :

-e Indicates that the next argument is an editor command.

-f Indicates that the next argument is a filename; the file contains editor commands, one to a line. Commands which are inherently multi-line, like a or c, should be written with the interior newlines preceded by `\'. Append mode is terminated by an unhidden newline.

The **-e** and **-f** flags may be intermixed in any order. If no **-e** or **-f** flags are given, the first argument is taken by default to be an editor command.

Addresses are allowed. The meaning of two addresses is: "Attempt this command on the first line matching the first address, and on all subsequent lines until the next line containing a match of the second address; then begin watching for a match of the first address and iterate." One address means: "Attempt this command on all the lines which match the address." allowable as addresses. A line number matches the cumulative line number of the input files. A `$' as an address matches the last line of the last input file.

-g Indicates that all s commands should be executed as though followed by a g. If only some substitutes are to be done globally, leave out the **-g** flag, and put the g's at the end of the appropriate command lines.

-n Prevents the copying of lines to the output by default. Only lines which are explicitly printed by p commands are written. In order to avoid getting double copies of some lines, the p command is a no-op unless the **-n** flag is set.

The intention is to simulate the editor as exactly as possible, but the line-at-a-time operation makes certain differences una-

voidable or desirable:

1. There is no notation '.'; and no relative addressing.

2. Commands with no addresses are defaulted to 1,$ rather than to dot.

3. Context addresses must be delimited by '/'; '?' is an error.

4. Expressions in addresses are not allowed.

5. Commands may have only as many addresses as they can use. That is, no command may have more than two addresses; the a, i, r, and l commands may have only one address.

6. A p at the end of a command only works with the s command. For other commands, or if the -n flag is not in effect, a p at the end of a command line is a no-op.

7. A w may appear at the end of a substitute command. It should be followed by a single space and a file name. If the substitute command is successfully executed, the line is written to the file. All files are opened when the commands are being compiled, and closed when the program terminates. Only ten different file names may appear in w commands in a single run. Unlike p, w takes effect regardless of the -n flag. If both p and w are appended to the same substitute command, they must be in the order pw.

8. The only commands available are a, c, d, i, s, p, q, r, w, g, v, and =. A successful execution of a q command causes the current line to be written out if it should be, and execution terminated. When a line is deleted by a d or c command, no further commands are attempted on the corpse, but another line is immediately read from the input (but see the next point (9)).

9. If an a, i, or r command is successfully executed, the text is inserted into the file whether or not the line on which the match was made is later deleted or not. Thus the commands:

```
/b/a\
XXX
/b/,/c/d
```

applied to the file

```
a
b
c
d
```

will produce

```
a
XXX
d
```

on the output.

10. Text inserted in the output stream by the a, i, c or r commands is not scanned for any pattern matches, nor are any editor commands applied to it.

SYNTAX:

Blank lines, blanks, and tabs at the beginning of a line in the command file are completely ignored.

Commands may be grouped by curly braces. The opening brace must appear in the place where a function would ordinarily appear; the closing brace must appear on a line by itself (except, of course, for leading blanks or tabs).

If the first line of a command file has #n as its first two characters, the no-copy flag is set, as though the **-n** option had been given on the command line. The rest of the line is ignored; it may be used for a title or a comment.

PATTERN MATCH COMMANDS:

These three capital letter commands are intended to allow pattern matches across new-line characters in the input file.

N Next line is appended to the current line; the two lines are separated by a new-line character which may be matched by `\n' (see below).

D Delete up to and including first (leftmost) new-line in the current pattern space. If the pattern space becomes empty (the only new-line was at the end of the space), read another line from the input. In any case, begin the list of editing commands again from the beginning.

P Print up to and including the first new-line in the pattern space.

META-CHARACTERS:

\n Matches imbedded newlines in the pattern space.

\\ Matches '\'

\] Matches ']'

Examples:

The file /mnt/btl/dir contains telephone-book entries. Most are on a single line; double line entries are signaled by having the second line begin with a blank.

To print out all double-line entries:

```
sed -n
    N
    /\n/p
    D /mnt/btl/dir
```

To combine all double line entries into single lines:

```
sed -n
    N
    /\n/{
        s// /p
        j (see flow-of-control commands)
    }
    P
    D /mnt/btl/dir >newdir
```

NEXT LINE COMMAND:

n The current line is replaced by the next line from the input file, and the list of editing functions is continued after the n function.

FLOW-OF-CONTROL COMMANDS:

These commands do no editing on the input line, but serve to control the order in which multipule editing commands are applied to an input line.

:<label>

The label command marks a place in the list of editing commands which may be referred to by j and t commands. The <label> may be any sequence of eight or fewer characters; if two different colon commands have identical labels, a compile time diagnostic will be generated, and no execution attempted.

j <label>

The jump command causes the sequence of editing commands being applied to the current input line to be restarted immediately after the place where a colon command with the same <label> was encountered. If no colon command with the same label can be found after all the editing commands have been compiled, a compile time diagnostic is produced, and no execution is at tempted.

A j command with no <label> is taken to be a jump to the end of the list of editing commands; whatever should be done with the current input line is done, and another input line is read; the list of editing commands is restarted from the beginning on the new line.

t <label>

The t command tests whether any successsful substitutions have been made on the current input line; if so, it jumps to <label>; if not, it does nothing. The flag which indicates that a successful substitution has been executed is reset by:

1) reading a new input line, and

2) executing a t command.

Example:

```
#n   This file prints each non-blank line following  a  blank
line.
1{
     /./p
}
/^$/{
: loop
     n
     /./{
          p
          j
     }
     j loop
}
```

FILES

SEE ALSO

ed (1)

DIAGNOSTICS

BUGS

Lines are silently truncated at 512 characters.

NAME

shlerr - shell error

SYNOPSIS

shlerr mode spcl etype ecode enum emsg

DESCRIPTION

Shlerr permits shell programs to call the sccerr or glberr subroutines to print error messages.

Shlerr has five or six arguments, depending on what value is specified for mode. A description of each argument follows.

mode is '0' if the error information is to be passed to subroutine sccerr.

is '1' if the error information is to be passed to subroutine glberr.

spcl is the address of a string containing the special characters associated with an error message. It is provided only if mode= '0'.

etype is the address of a string containing the type or severity of an error.

ecode is the address of a string containing the three-character error code.

enum is the address of a string containing the three-character error number.

emsg is the address of a string containing the message associated with an error.

FILES

SEE ALSO

sccerr(3L), glberr(3L)

DIAGNOSTICS

Value for mode must be '0' or '1'. If not then the following error message is printed:

?F USR 100 INVLAID ARGUMENT

NAME

submit - run a backround process

SYNOPSIS

submit [-p priority] command [arguments]

DESCRIPTION

Submit is based on the TSO submit command. It is used an execute an independent backround process. It is particularly useful if you wish to execute a long running process without tying up a terminal. Process output (including error output) is directed to a file named "sub#####.out". A file named submit.log is created to record start and finish times for submitted processes. Finally, if the user is signed-on when a submitted process completes (s)he receives a message on the terminal.

The user may specify a priority for a submitted process by including a -p flag followed by a number from 0 to 20. Minimum priority is 20 and maximum is 0. For further information see the "nice" command.

BUGS

Probably

NAME

spp - shell command file pre-processor

SYNOPSIS

spp file [args ...]

DESCRIPTION

Spp provides a subroutine facility for shell procedures. It can also be used to package a set of shell procedure files and data files into a single file.

File contains a set of shell commands interspersed with ``label'' lines. A ``label'' line commences with a ``name'' (up to fourteen lower-case alphabetic characters) followed immediately by a colon. Each such ``name'' may be used elsewhere in file as the name of a command (i.e., as a routine name). Spp creates a temporary directory and copies into it files obtained by splitting file at ``label'' boundaries. The first such file is called ``main'' and succeeding ones are named from the ``labels'' that precede them in file. Spp changes the value of **$PATH** so that the temporary directory is searched initially when command names are resolved and then executes the command ``args ...'', or, if this is null, ``main''.

EXAMPLE

If the file ``sample'' contains:

```
        for i in `ls`
        do show $i; done
show:   subroutine
        echo $1
old: free standing routine
        wc * ^ tail -1
```

the commands **spp sample** and **spp sample old** will display the contents of the current directory and its size, respectively. Note that the first command is equivalent to **spp sample main** and that everything that appears on a label line after the colon is treated as a comment and ignored.

FILES

/usr/tmp/spp$$	temporary directory
/usr/tmp/spp$$/...	temporary shell procedures
/usr/lib/breakup	program to rewrite command file

SEE ALSO

sh(1).

BUGS

Label names must consist of lower case alphabetics only. Spp will fail if **$PATH** is not exported properly by the shell. Occasionally, the temporary directory is not deleted.

NAME

tm - magnetic tape manipulate

SYNOPSIS

tm <filename>

DESCRIPTION

Tm can open, read, write and close <filename> on command. For more info type '?' after ':' prompt.

NAME

updsrc - installation or retrieval of code

SYNOPSIS

```
updsrc      [+o,+a,+c,+d,+gv]
            ca=<ca_number(s)>
            ma=<resolved_by_name>
            pr=<pr_directory>
            ma=<make_name>
            <file(s)-and/or-au(s)>
```

DESCRIPTION

Updsrc is twofold in nature. First, it is a special routine for Generic Administration (GA) housekeeping of the update packs. It will be used to allow GA to control what code is on the packs. Also, it will provide correlation between all code on the update packs and the associated Change Administrations (CAs). All code to be made on the <make_name> must be submitted via updsrc. All CA(s) to be signed-off for all code in <make_name>, must be submitted via updsrc.

Second, it provides the user with a retrieval of the latest code on the gensrc and update packs to update their local directory.

When invoking updsrc the user has three main options of command line usage:

1 - Type the command name alone.
EFFECT: The SYNTAX line will be returned.

2 - Type the command name and any one or more of its arguments in any non-specified order.
EFFECT: The user will be prompted for any missing arguments, except for optional arguments denoted in SYNOPSIS section of this IM.

3 - Type the command name and all necessary arguments in any non-specified order.
EFFECT: Updsrc will execute according to the specified arguments and will not prompt anything more for the command line.

The command line is totally non-positional. The user may place in any order the four keywords and their associated arguments. The options can be placed in any point of the string of arguments. And filenames and/or au names can be placed anywhere between and/or around the other arguments.

The <pr_directory>, <file(s)-and/or-au(s)> and <make_name> arguments will be checked and the code will be sent to the appropiate update pack via putsrc; or they will be checked

and the code retrieved via getsrc, depending on the options used. Unless the '+g' type option is used, specifing the retrieval option, the default is to update the update pack specified by <make_name>.

The arguments to updsrc are as follows:

OPTION	EFFECT
+o	If <file(s)-and/or-au(s)> exist in the target directory, overwrite them with the new files and inform user. If this option is not set, the routine will prompt the user if the existing files are to be over-written. CAUTION: Do not use this option unless sure of what is being over-written; the over-written source will be gone and may only be retrievable via tape dumps.
+a	If this option is specified, the au file names specified will be opened and the #PROGRAM UNITS and the #DATA UNITS sections will be read. All the source files and header files listed in those sections will be executed upon also.
+c	When this option is used, updsrc will not copy <file(s)-and/or-au(s)> to the specified location, but instead will be used to record any <CA_numbers> for the <make_name> for <file(s)-and/or-au(s)> that the user forgot to input previously when the <file(s)-and/or-au(s)> were first brought over via updsrc.
+d	(Delete option not available at this time.)
+gv	The '+g' or '+gv' options cause updsrc to retrieve the latest copies of the <file(s)-and/or-au(s)> off the gensrc and/or update packs. The code is deposited in your present working directory. The usage of 'v' with the '+g' option turns on the verbose mode. This

can be very helpful in finding out the status of all the update packs and gensrc concerning where your code resides.

ARGUMENT	DEFINE
<file(s)-and-au(s)>	The name or names of source and/or the name or names of au files to be updated. If an au file name and the '+a' option is used, updsrc will not only copy the au files over, but will also read the #PROGRAM UNITS and the #DATA UNITS sections of the au and will copy over all the source and header files listed there. (Note: '*' is allowed in the command line for shell expansion of source file names or au file names in the local directory, but will be rejected if used to expand names when prompting option is used.)
<pr_directory>	The update pack directory where code is to reside, or currently resides.
<make_name>	The appropiate generic and issue in which the source is to be made. (i.e. SC7I1, SC6I4, etc.)
<ca_number(s)>	Multiple CA numbers should be separated by spaces. This argument is not necessary and will not be prompted for: When '+g' type option is used or if <pr_directory> is a new pr as defined by GA.
<resolved_by_name>	The name of the CA 'RESOLVED BY' person. This argument is not necessary and will not be prompted for: When '+g' type option is used or if <pr_directory> is a new pr as defined by GA.

In terms of putting code on update packs: All elements entered into updsrc will be checked for complete correlation. After the user has answered the all prompts, updsrc returns the user to the UNIX enviroment and spins off the backround check. If there is any problem in updsrc's check of given data, the user will receive mail concerning the

error. And, if there are no problems, the user still will recieve mail confirming the code's acceptance by GA.

In terms of retrieving code: Updsrc will process the request on line.

SEE ALSO

putsrc(3L), getsrc(3L)

NAME

userlist - generates **SCCS USERS REGISTER** from password file(s) supplied as argument(s)

SYNOPSIS

userlist passwordfile1 [passwordfile2 ...]

DESCRIPTION

Userlist extracts and formats a sorted list of SCCS Laboratory Users from the named password file(s) and sends the list to the standard output. **Userlist** expects the comment fields of the password file entries to follow a prescribed format as described under **PASSWORD FILE FORMAT.**

PASSWORD FILE FORMAT

Userlist extracts the **SCCS USERS REGISTER** from the GCOS fields of the password file entries. In the SCCS Lab the GCOS field is used as a comment field and will be referred to as the comment field in the remainder of this description.

For **userlist** to function properly, the comment field of each password file entry is divided into 4 or 5 subfields separated by commas. Five subfields are required if a user priority is included as described in passwd(5); otherwise, only four subfields are required. Most SCCS password file entries have four subfields called name, telephone, location, and department. The telephone and/or location subfield may be left blank, but the terminating comma for each missing subfield must appear.

The four subfields must conform to the following conventions or an appropriate diagnostic will be issued to the user's standard error output:

name

This subfield contains the user's first and last names, first name first. The name included in the subfield may consist of more than two words, but the second word is always assumed to be the last name for sorting purposes. If the name subfield is too long, **userlist** truncates it to 23 characters. A legal character in the name subfield is any of the upper or lower case letters, a blank, or a period.

Both the first and last names must begin with capital letters. If the first name begins with a lower case letter, **userlist** ignores that password file entry, and that user will not appear on the **SCCS USERS REGISTER** unless he/she has a subsequent password file entry with an appropriately formatted comment field.

telephone

The telephone subfield has a maximum length of 15 characters

which may be any of the following:

0-9, -, (,), x, blank

A password file entry having a telephone number longer than 15 characters will be rejected.

location

The location subfield has a maximum length of 12 characters, the only illegal character being a comma. BTL locations should be of the form **XX-SHFYY** where,

XX = two-letter location code
S = building section number
H = hall letter
F = floor number
YY = room number

department

Users are divided into two categories -- direct and sponsored. Direct users are those employees currently reporting to a BTL supervisor at Columbus. Sponsored users are WECo employees, Telco employees, and BTL employees at other BTL locations. Each sponsored user has a sponsor who is a direct user. **Userlist** uses the department subfield to distinguish between direct and sponsored users.

The department subfield for a direct user consists of a capital D followed by a 4-digit department number. The department subfield for a sponsored user consists of the login id of the user's sponsor in lower case letters.

The department subfield must contain from 1 to 5 non-blank characters.

DIAGNOSTICS

Userlist issues two kinds of diagnostics to the standard error output -- fatal and non-fatal errors. A fatal error will occur if any of the following conditions are encountered:

1. No password file names were supplied as arguments.

2. The number of users found in the password file(s) exceeded 150.

3. **Userlist** could not open one of the named password files for reading.

A fatal error will cause **userlist** to issue an appropriate error message and to exit with a status of minus one (-1).

A non-fatal error will occur if the comment field cannot be extracted from a password file entry or if one of the formatting conventions described under **PASSWORD FILE FORMAT** is violated. If such an error occurs, the user will not appear on the **SCCS USERS REGISTER** unless **userlist** finds a subsequent password file entry for that user with an appropriately formatted comment field.

Userlist assumes that the first valid comment field found for a particular user is correct and uses that comment field to build his/her entry in the **SCCS USERS REGISTER.** If a subsequent comment field is encountered which differs from the first one found, an appropriate non-fatal error message will be issued.

SEE ALSO

passwd(5)

BUGS

Userlist sorts first by department number and then by last name. No sorting is done by first name. People with the same last name and different first names appear in random order within their department and last name grouping.

NAME

read, open, onend - sequential file read

SYNOPSIS

onend [label]
open [fname]
read

DESCRIPTION

Within a command file, read accesses data, one line at a time from another file. The read is done sequentially from the beginning of the file.

Open is needed to identify the file to be read. Only one file may be open at any time. Without an argument the standard input is used.

Onend provides a branch to label when the EOF condition is reached on fname. If label is not supplied, a diagnostic is written to the error output,and a null string is sent to the standard output.

FILES

SEE ALSO

sh (1), goto (1), read (1)

DIAGNOSTICS

BUGS

Standard Shell syntax is violated to allow pipe and redirected input to take precedence over an open file, which in turn takes precedence over standard Shell input.

Don't try to read a directory file.

Bell Laboratories

SUBJECT: NEW MANUAL PAGE FOR XPR DATE: 11/15/82

FROM: J. E. Defenderfer

Attached is the new manual page for **xpr** which produces printed listings and/or cross references (x-refs) of source files. The new version is on the development machines.

In summary, the changes to **xpr** are:

- A new option, -d, which produces x-refs of "definitions" (i.e. defined symbols, structure and union definitions, and typedef's). A definition x-ref is particularly useful for header file listings.
- New options, -moO, which facilitate listings intended for double sided copying (e.g. UNIX opsys listings).
- A new option, -j, which causes file names rather than file numbers to appear in the x-refs. This is most useful when you produce listings via another program like **spr**, and you also want the x-refs of **xpr**.
- Improvements in the heuristic parsing of C files to reflect new C features and remove selected deficiencies.

Some Notes on General Use. A few comments on the three options of **xpr** which are frequently used, **-w<wid>**, **-!**, and **-h**:

- The **-!** option produces all three x-refs. The best x-refs are produced when headers precede other source files in the listings.
- For code without function prologues (frequent in UNIX code and less frequent in SCCS code), the **-h** option helps visually identify the start of functions.
- **xpr**, with a default line width (72 characters), produces a listing which is ideal for 8.5"x11" sheets for notebook insertion. Unfortunately, lots of code is very wide, and so many lines are folded that the listing is crummy. Thus, an option of **-w80** or **-w100** is often necessary. I avoid the problem by writing narrow code which easily fits in program notebooks. Code which is barely too wide for **xpr**'s defaults, may fit notebooks without excessive line folding by manipulating several options simultaneously; for example, dropping the line numbers using **-n**, increasing the width, and/or decreasing the left margin with **-c<page offset>**.

Your comments and suggestions are welcome.

Copy to:
All Members of Depts.
59472 and 59475

SIGNED Joe ORG. 59472

LOCATION CB 2B208 EXTENSION x4507

THIS FORM IS DESIGNED FOR INFORMAL HANDWRITTEN MEMORANDA.

E-990-B (1-74)

NAME

xpr - Make listing and cross reference of C program

USAGE

xpr [-bc<col>dfFghijl<lines>mnoOsuvw<wid>x<num>] file file ....

DESCRIPTION

Produces printed listing and/or cross references of one or more files. Each page is headed with the current date, the file name, the file number, the page, and the first line number of the file on the page. By default, printed lines are prefixed with line numbers. Lines over 72 characters long are folded at a blank, tab, comma, semicolon, closing bracket, brace, or parenthesis; lines without any of the seven fold characters are not folded. The "overflow" text of a folded line is prefixed with the leading white space of the original line, and it is not truncated again even if it very long. Optionally, each function definition is highlighted.

Optionally, cross reference lists may be generated for functions, external variables, and definitions. In the cross reference lists, each entry includes the function of the reference, the line number of the reference, and the line of code. Options:

-b Print nesting level on each line (default = OFF).

-c<col> Column offset of output (default = 4).

-d Definition cross reference (default = OFF).

-f Function cross reference (default = OFF).

-F Function definition listing (default = OFF).

-g Global cross references (across all input files) (default = OFF).

-h Highlight function definitions (default = OFF).

-i Inhibit program listing (default = OFF).

-j File numbers in cross references are replaced by file names (default = OFF).

-l<lines> Number of lines in a page (default = 66).

-m Page headers are "mirrored" for even pages which is useful for double sided printing (default = OFF).

-n Print lines numbers on each line (default = ON).

-o Skip to odd page before starting next file (default =

OFF).

-O Skip to odd pages before all files (default = OFF).

-s Substitute the next argument as the printed name of the next file processed.

-v Variable cross reference (default = OFF).

-w<wid> Width of printed lines, which does not include the column offset, line numbering, or nesting level (default = 72).

-x<num> Set file number of next processed file.

-! Selects the set of options, "gfFvd", which opts for all cross references and each to be global.

xpr will read the environmental parameter, XPR, as a set of options to the program. Thus, you may reset any of the defaults listed above. Each invocation of an OFF/ON flag toggles that flag. You may put options between file names to change the treatment of subsequent files. If you call **xpr** with no arguments, then it tells you how to use it. If you call it with an argument of just "-", then standard input is read as a file.

FILES

/tmp/FcxXXXXXX - Temp sort file for function x-ref
/tmp/VcxXXXXXX - Temp sort file for variable x-ref
/tmp/DcxXXXXXX - Temp sort file for definition x-ref

BUGS

Uses heuristics to produce the cross reference; in particular, "defines" can be used in such a way as to really screw the cross references up. References to external variables are not collected until the program encounters an external declaration for that variable; thus, the ordering of the files may cause a difference on the variable cross reference although the function cross reference is often correct regardless of the ordering.