Release of Patchy version 5

Julius Zoll

October 5, 1994

The 3 major steps forward taken with this version 5 are:

1. it is now possible to freely change a Pam file with an editor, constructing later, if needed, an update cradle by comparing the final against the original Pam file using the new Patchy auxiliary Nydiff.
2. in order that such a cradle constructed with Nydiff may be completely general it is now possible to insert "delayed control-lines" under control of a `+ADD` or `+REPLACE` to become operational when the insertion target is processed.
3. the new Patchy auxiliary Nyshell can be used on Unix machines and on the VAX to automatically select for re-compilation all routines whose source or include files have changed, for a set of routines which together make up a library for one package.

This paper is a conversion guide, and also an update on the existing Patchy documentation until the reference manual is updated. It contains the following sections:

1 Program call statements

The new programs of version 5 no longer handle compact binary Pam files, but only ordinary text files ("car" files). Thus versions 4 and 5 have to co-exist for some time; the program calls for version 5 are:

Replacing ypatchy:

```
nypatchy pam fort cradle print cc as data  fort:2 cc:2 as:2 data:2
```

Replacing the auxiliaries yindex and ylist:

```
nyindex  pam options print
nylist   pam options print
```

The new auxiliary Nydiff compares two Pam files to construct a cradle which can be used to update the "old" Pam file to the state of "new":

```
nydiff   old new diff options print
```

The new auxiliary Nycheck checks all Patchy control-lines on a Pam file for correct syntax:

```
nycheck  pam options print
```

The new auxiliary Nyshell can be used to replace fcasplit:

```
nyshell  log options cradle print
```

Help about the auxiliaries can be obtained by typing for example:

```
       nydiff --- H
   or  nydiff help
```

In the first case the option letter H asks for help about the auxiliary itself; the second case asks for help about the syntax of the parameters.

2 Overview of the differences of version 5 to 4

2.1) Note that on the command-line `nypatchy` the order of the ASM output files has been rationalized and differs from the order of `ypatchy`:

```
 ypatchy pam fort cradle print as data fort:2 as:2   data:2 cc    cc:2
nypatchy pam fort cradle print cc as   data   fort:2 cc:2   as:2 data:2
     (p= 1   2    3      4     5  6    7      8      9      10   11)
```

2.2) On the IBM with VM/CMS calling the programs into action is now just like on UNIX or VAX, including the dialogue feature, see sect. 8.

2.3) File names are now case-sensitive; environment variables contained in file names are now specified Unix-style as `${name}` or maybe `$name`, but the curly brackets are required on non-Unix machines. The defaults for "cradle" and "print" are now standard input and output. To get details, type `HELP` in response to the prompt from any of the programs.

2.4) File names may be given in Unix-style on the VAX; on the IBM file names are given either as `fname.ftype.fmode` or as `fmode/fname.ftype`.

2.5) Status return codes from the programs are:

UNIX	VAX	IBM	
$status = 0	$STATUS = 1	rc = 0	normal
1	9	1	no error, but void operation
2	4	8	error

For Nypatchy the status must be tested, it stops delivering ASM output if an error occurs.

2.6) Pam files are now plain text files; the compact binary format of version 4 is no longer handled.

2.7) The specification for the control-line `+ASM` has changed substantially, see sect. 3 and sect. 7.

2.8) Control-line `+PAM` has changed a bit, see sect. 3.

2.9) With the new control-line `+MORE .file-name` given in the blank deck of the cradle, the cradle can be continued with material from another file.

2.10) For any control-line given <u>in the cradle</u> substitution of an environment variable can be requested by giving the name of the variable enclosed by curly brackets and preceded with the `$` sign, for example:

```
+USE, *PATCHY, ${PLINAME}.
```

will call for replacement of the string `${PLINAME}` by the contents of the environment variable `PLINAME` before the line is cracked for syntax. Thus one could have a shell commmand like

```
setenv PLINAME "CONVEX, 64BIT"
```

Note that this cannot be used for c/lines on Pam files, nor on delayed control-lines.

2.11) "Delayed control-lines" is the main new feature of version 5. It is now possible to drive by means of a `+ADD` or `+REPLACE` Patchy control lines to a target, to be interpreted when the target deck is processed. For example:

```
+ADD, *WYLSUN,, 7.
-USE, WINDOWS.

+REPL, WCDE,, 10-14.
-KEEP, CONSTLD, IF=LEAD11.
        PARAMETER   (NCHXBL=0, IQTYPE=6, NQL=512)
-KEEP, CONSTLD, IF=LEAD12.
        PARAMETER   (NCHXBL=1, IQTYPE=6, NQL=512)
```

Delayed c/ls are marked by "-" in column 1, to be converted to "+".

2.12) Running in update mode has now to be signalled by the new control line `+UPDATE` in the blank deck of the cradle, see sect. 4; sect. 9 shows an example.

2.13) The title of a Pam file, that is any material before the first line `+PATCH,...`, is now handled slightly differently: it is considered as a patch whose name is derived from the Pam-file identifier; see the description of `+PAM` in sect. 3.

2.14) The old line-length limit of 80 characters has been increased to 512. Listings are "nice" for lines up to 80 characters, and "ugly" for longer lines. Control-lines are no longer cut after column 72.

2.15) The old limit of 8 characters maximum for patch/deck/sequence names has been increased to 32. Listings are "nice" for names of up to 12 characters. For back-compatible processing of Pam files one can limit the number of significant characters in names to 8 with the new c/l `+NAMES,8.` given in the blank deck of the cradle.

2.16) For some control-lines the syntax checking is now more rigorous. In particular: of the values given to the "T=" parameter only the first letter is significant with version 4; with version 5 subsequent characters are also significant, if given. For example:

```
+OPTION, COMPACT, OFF.
```

is fine, but

```
+OPTION, COMPACT, ON.
```

is now failed. Sect. 3 gives details per command.

2.17) With version 4 the truth-value for an IF= parameter is evaluated at the moment when the control-line is read. With version 5 this evaluation occurs only when the c/l is used; thus for example:

```
+ADD, WYLBUR, XSET_RAN, 7.
+CDE, WWC, IF=WINDOWS.
```

given in the cradle will now work in the way one would like it to.

2.18) There are no more "lost activations" with version 5, which has the complete deck in memory before it is processed (cf Patchy Ref.Man. 4.22).

2.19) Version 4 tried to be too subtle in deriving processing mode selection from patches quoted in IF= parameters. It is now simply the logical OR of the attached EXE bits of all patches quoted.

2.20) The processing-mode filter for c/l `+DIVERT` has been taken away; thus one can specify on the Pam file which routines are to be compiled without optimisation under which circumstances, without one having to activate this from the cradle (cf Patchy Reference Manual 4.05).

2.21) The new Patchy initialises the patch-name `PY_VS5` to be "used". This can be useful when one wants to make a compatible Pam file or cradle usable with version 4 and 5.

2.22) For the maintenance of complete programs or packages the philosophy of the use of Patchy has changed substantially: Using the combination of the new Nypatchy and the new Nyshell one can automatically re-compile only those routines whose source code has changed since the last successful compilation. To achieve this, Nypatchy has been programmed to deliver, on demand, its ASM output split onto individual files, one for each deck. If the ASM output from the previous run still exists, Nypatchy can be instructed to read this back into memory and to compare it to the expected new output. If the two are identical, the file is not re-written and thus its time-stamp is not changed. Nyshell receives from Nypatchy a list of all the routines in the package and it creates a shell-script to compile all the routines which need to be re-compiled. For details see sect. 7 and sect. 10.

3 Changes in specifications per control-line

3.1) `+DECK, dname, T=type, IF=...`

with the `T=` parameter one can signal the data-type of the contents of the deck. The standard data types are now:

```
T=FORT   for Fortran           (also T=CO*mpile)
T=CC     for C language        (also T=X*cc)
T=AS     for assembler         (also T=A*s*)
T=DATA   for data              (also T=D*ata)
T=INCL   for C include files
T=SHELL  for shell-scripts
T=CRAD   for cradles
```

The right-hand set is for back-compatibility, the `*` separates the required from the optional extra characters. `A*s*` means A or anything starting with AS.

It is now possible to have user-defined data types; this is described in sect. 7.

The special T parameter `J*oin` is still used to suppress page-ejects in listings at the start of the deck which has it.

3.2) `+PATCH, pname, T=type, IF=...`

as for `+DECK`, with the extra T parameter `R*epeat` as in version 4.

3.3) `+ASM, ...`

The handling of ASM output had to be thoroughly re-organized to be able to handle the user-defined data types mentioned above. As a result the line `+ASM` is now quite different, see sect. 7. Nypatchy has been programmed to recognize the simpler old forms of `+ASM`, here comes a summary, for details see sect. 7.

Lines like

```
+ASM, 31, 34.
+ASM, 31, 34, T=ATT   .fcdiv.f
```

are handled correctly.

Lines like

```
+ASM, 31.
+ASM, 24, 34.
```

are accepted, but are redundant: giving a file-name on the program call statement for a particular ASM stream establishes this stream; stream 34 (CC diverted) is linked to stream 24 (CC normal) by default.

A line like

```
    +ASM, 23, R=!           ./*DECK ID>, !.inc */
or  +ASM, 23, LUN=21, R=!   ./*DECK ID>, !.inc */
```

will cause an error, normally one would replace it by

```
+ASM, DATA, T=RH  .!/*DECK ID>, !.inc */
```

but if one wants to use the same cradle with versions 4 and 5 for a time one could give:

```
+ASM, 23, R=!, IF=-PY_VS5   ./*DECK ID>, !.inc */
+ASM, 23, T=RH, IF=PY_VS5  .!/*DECK ID>, !.inc */
```

A line like

```
+ASM, 23, LUN=24, R=!   ./*DECK ID>, !.inc */
```

would have to be replaced by 2 lines

```
+ASM, DATA, T=RH  .!/*DECK ID>, !.inc */
+ASM, CC, DATA, T=BIND.
```

the second line binding stream 23 (DATA) to stream 24 (CC).

3.4) `+OPTION, opt1, opt2, ..., IF=...`

At least 3 characters have to be given for the "opt" values, and further characters, if any, have to be correct.

```
OFF        as before, set options OFF, not ON;
ALL        as before;
BAC*kcom   back-compatible mode, this relaxes the new syntax
           checks somewhat, "on" by default for the time being;
COM*pact   as before, "on" by default;
EJE*ct     new: honor c/ls "+SEQ, QEJECT, N=n" in listings;
FUL*L      as before, "on" by default;
MAP*asm    as before;
VER*bose   new: print full summary at the end,
           "on" by default only if the printed output goes
           to a file, not to the screen.
```

3.5) `+PAM, (LABEL=)n, N=x,y, T=opt, RETURN=pname, IF=...  .file-name`

All of these parameters are optional, their significance is much the same as with version 4:

```
LABEL=n    with n a small integer, this is needed only to identify
           a PAM to be read partially and to be resumed later.

N=ndo      as before: process the first ndo PAM files, default 999
N=nsk,ndo  skip the first nsk PAM files, and then process
           the next ndo files.

T=opt:
T=ATTACH   as before: read the file whose name is given in the
           comment field; the LUN=lun parameter is no longer needed;
T=HOLD     as before: no rewind/close at the end of processing;
T=RESUME   as before: no rewind before processing;
T=UPDATE   as before: PAM file to be updated;
T=CARDS    is accepted and ignored for back-compatibility.

R=pname    as before: stop reading after patch pname.
```

The old parameter `P=pname,type` is no longer needed nor accepted: the title is now considered to be a patch whose name is the Pam-file identifier prefixed with the "commercial at", thus the patch name of the title of the ZEBRA Pam file would be `@ZEBRA`.

The typical old `+PAM` control-line like

```
+PAM, 11, T=CAR,ATTACH   .zebra
```

is still fine, the "11" is now taken as a label which in this case is not useful but does not hurt, the CAR option is ignored because Pam files are now anyway in "card" format, it has the same effect as the new standard:

```
+PAM, T=ATTACH .zebra
```

3.6) `+PARAMETER, option, N=value, IF=...`

```
LIN*es     simplified, give only one value to "N=",
           the number of lines per page, default 110
COL*umns   new: the page-width, default 120, minimum 90
CLA*sh     as before
```

3.7) `+TITLE`

One can now give the Pam file title directly on the `+TITLE` line:

```
old:   +TITLE.
       PATCHY   5.00 /55  1994/02/08 11.20

new:   +TITLE:  PATCHY   5.00 /55  1994/02/08 11.20
```

Note the use of the colon `":"`.

4 New control-lines for version 5

4.1) `+NAME, length, nslots, nbytes`

Giving this c/l in the blank deck of the cradle one can control the name stack of Patchy:

```
length  the number of significant characters in names, default 32
slots   maximum number of names, default 2000
nbytes  number of bytes of text store reserved, default 24000
```

Giving `+NAMES,8.` one gets back-compatibility in the handling of long names.

4.2) `+GAP, gc, gl, maxl, chpl`

Giving this c/l in the blank deck of the cradle one can control the space limits at which garbage collection occurs.

```
gc      the number of free words in the control store
            default 4000, minimum 1000
gl      the number of free line slots
            default 200, minimum 100
maxl    the maximum number of lines expected in any deck
            default 5000, minimum 2000
chpl    the mean number of characters per line
            default 40, minimum 20
```

4.3) `+MORE .file-name`

With this control-line one can continue the blank deck of the cradle from the file whose name is given in the comment field.

4.4) `+UPDATE.`

For an update run this control-line must be given in the blank deck of the cradle. The old options `KEEPORD, SEQORD, YESIF` are no longer needed nor accepted. Delayed control-lines, marked by `"-"` instead of a `"+"`, will be inserted and upgraded. Wether a Pam file contains correction material or is a Pam file to be updated is signalled by the `T=UPDATE` parameter on the c/l `+PAM` as before.

4.5) `+SKIP, IF=...` is the opposite of `+SELF, IF=...`, that is the material following is ignored if the condition is "true". In particular `+SKIP.` is an unconditional skip.

4.6) +_IF, ... et al. to mark conditional material:

```
+_IF, [IF=]...      open an IF section of material, accept the
                    material just following if "true"

+_ELSE, [IF=]...    accept if the initial +IF and all previous +ELSE
                    have failed, and if the IF condition is true.

+_ENDIF             terminate an IF section.

+_IFNOT, ...        is the opposite of "+_IF, ...", ie. the material
                    is accepted if the condition is "false".
```

These statements may be nested, in which case the number of underscores should indicate the nesting level. Level 1 may be indicated either by zero or by 1 underscore, level 2 should have one more, etc.

These are "soft" control-lines, ie. they may occur within actions or `+KEEP`'s.

4.7) `+NIL.` is a comment line, also a "soft" control-line.

For many comment lines in succession it is better to head them with `+SKIP.` and terminate with `+SELF.` or some other "hard" c/l.

4.8) `+SHOW, ASM.`

This will display the current state of the ASM handling control structure. This may help if one is in doubt what Patchy did of one's `+ASM` lines. If given in the blank deck of the cradle one sees the structure before defaults have been resolved (defaults are marked by -1), an operation which is executed only at the end of the blank deck of the cradle.

4.9) `+XDIVERT, pname [,D=dname].`

Extra diversion is like the old diversion triggered by `+DIVERT` at one level up. See sect. 7 for details.

4.10) `+INCLUDE, NAME.`

This causes insertion of the line

```
#include "name.h"
```

into the ASM output at this point, in lower case, and it signals to Patchy that the current deck depends on this include file, a piece of information passed on to Nyshell via the log file. See sect. 5 for further details.

5 Handling of include files

An include file present on the user's Pam file and called from some of the decks on this Pam file is defined in a deck of its own with data type INCL, for example:

```
+DECK, MZINC, T=INCL.
text of the include file
```

and its use is signalled with the control line

```
+INCLUDE, MZINC.
```

which line is replaced by Nypatchy with

```
#include "mzinc.h"
```

The purpose of this construction is to make Nypatchy/Nyshell aware of the dependency of decks on include files, to cause re-compilation of all decks which depend on include files which have changed.

If the ASM output is generated in MODIFY mode, Nypatchy itself knows whether an include file has changed and marks itself dependent decks for re-compilation. At the next step, Nyshell will check on the dates of the object files and dependent include files of all decks which Nypatchy has signalled as unchanged to make sure that re-compilation is really not needed.

6 New built-in sequences

The new sequences **+QFxxxx** get components of the Pam file header line with this general syntax:

```
+SEQ, QFxxxx, S=sym, L=lim, N=n  .txa?txe
```

Such a sequence call will expand into one line of text starting with the text "txa" given in the comment field, with leading blanks significant, followed by the "value" substituted for the escape symbol (here and by default `"?"`), followed by the text "txe" (if any).

```
With S=sym the escape symbol can be specified, default "?".

With L=lim the length of the value string can be limited,
default is the 'natural length' of the item, which for QFHEAD
includes the file ID, the version, date and time of the file.

With N=n the ordinal number of the Pam file whose header line
is to be used can be selected, default: the current Pam file.
```

6.1) Get the Pam file header line as is:

```
+SEQ, QFHEAD, ...

    Example: +SEQ, QFHEAD  .      DATA VIDQQ /'@(#)?>'/
 might give:       DATA VIDQQ /'@(#)WYLBUR  1.08 /7  920229 12.00>'/
```

6.2) Get the Pam file identifier:

```
+SEQ, QFNAME, ...

    Example: +SEQ, QFNAME  .      CHID = '?'
 might give:       CHID = 'WYLBUR'
```

6.3) Get the version as a text string:

```
+SEQ, QFVERS, ...

    Example: +SEQ, QFVERS  .      CHVERS = '?'
 might give:       CHVERS = '1.08 /7'
```

6.4) Get the version as a number:

```
+SEQ, QFVSNUM, ...

    Example: +SEQ, QFVSNUM  .      int pamvers = ?;
 might give:       int pamvers = 10807;
```

6.5) Get the primary version number:

```
+SEQ, QFVPRIM, ...

    Example: +SEQ, QFVPRIM, S=@  .      NUMVERS = @ * 100.01
 might give:       NUMVERS = 1.08 * 100.01
```

6.6) Get the secondary version number:

```
+SEQ, QFVSEC, ...

    Example: +SEQ, QFVSEC, S=@  .      NUMALL = 1000*NUMVERS + @
 might give:       NUMALL = 1000*NUMVERS + 7
```

6.7) In the same spirit, the new sequences `QDATE` and `QTIME` give the date and the time of the Patchy run not neccesarily tied to Fortran:

```
+SEQ, QDATE, S=sym  .txa?txe

           Example: +SEQ, QDATE   .      PARAMETER (IPDATE=?)
        might give:       PARAMETER (IPDATE=19920229)
```

```
+SEQ, QTIME, S=sym  .txa?txe

           Example: +SEQ, QTIME   .#define now ?
        might give: #define now 2204
```

6.8) The new sequence `QENVIR` calls for substitution of an environment variable into the source statement given:

```
+SEQ, QENVIR  .statement

           Example: +SEQ, QENVIR  .      PARAMETER (DIR='${DIR}/')
        might give:       PARAMETER (DIR='/cern/dev/')
```

7 Data types and ASM ouput handling

On the Pam files the authors indicate the data type of the material contained in each patch or deck. The data type gives a generic handle to specify in the cradle what to do with the material of each type.

"Diversion" is primarily an instrument to separate routines which have to be compiled with down-graded optimisation from the main-stream routines. "Extra diversion" can be used to further split the material, for example into the routines of a program and the routines for testing the program.

The combination of data type plus diversion results in what is called a "logical (output) stream". With the c/l `+ASM` given in the blank deck of the cradle one can overrule Nypatchy's default assumption on what to to with the material of each logical stream.

7.1) **Data types**

Data types are established by the material on the Pam file. Any deck or patch can indicate its data type explicitely with the `T=` parameter;

```
defaults: for a deck:   type of its patch;
          for a patch:  Fortran.
```

Standard data types

There are 7 standard data types:

```
T=FORT     for Fortran
T=CC       for C
T=INCL     for C include files
T=AS       for assembler
T=DATA     for data with no implied special actions
T=SHELL    for shell scripts
T=CRAD     for cradles
```

(The old keys words `CO[mpile]`, `X[cc]`, `AS[...]`, `D[ata]` are supported for back compatibility. The square brackets indicate optional additional characters.)

User data types

There may be any reasonable number of user data types on the Pam file(s). For each type the user chooses a name and sticks to it (no abbreviations), For example:

```
+PATCH, DOC, T=LATEX.
```

The standard data types are reserved names, and also `J[oin]`, `R[epeat]`, `CO[mpile]`, `X[cc]`, `AS[...]`, `D[ata]`.

Back-compatibility of Pam files

To use the new data types on Pam files which may still have to be read with Patchy 4, one may declare an old data type followed by a new one, for example:

```
+DECK, COUM, T=DATA, SHELL.
```

Patchy 4 will see "DATA", whilst Patchy 5 will see "SHELL".

Or:

```
+DECK, AQUA, T=XCC, INCL.
```

Patchy 4 will see XCC, Patchy 5 will see an include file and therefor add the extension ".h" by default.

Careful: do not use a user data type starting with any of the letters A, C, D, J, R, X if you want to be back-compatible.

7.2) Logical streams and Diversion

Material may be diverted or extra-diverted with the control lines

```
+DIVERT, pname[, D=dname].
+XDIV,   pname[, D=dname].
```

The logical streams are identified by the name of the data type and a suffix, for example:

```
LATEX:1   normal/normal
LATEX:2   normal/diverted
LATEX:3   extra/normal
LATEX:4   extra/diverted    stream of data type LATEX
```

The generic logical stream LATEX (or LATEX:0) designates all logical streams of the this data type.

7.3) Physical streams and Output files

A logical stream may be made physical by giving output instructions for it, a file name in the simplest case. Several logical streams may be bound to a physical stream.

To establish a physical stream one may give in the cradle one of the 4 forms of the control-line +ASM (using FORT:0 as example):

```
    +ASM, FORT, T=ATTACH   .fname
or  +ASM, FORT, T=SPLIT.
or  +ASM, FORT, T=MODIFY.
or  +ASM, FORT, T=BYPASS.
```

These examples each establish the physical stream FORT. In the first case the file name is given on the control-line, in the second and third case each deck is written to a separate file (see later), in the last case all material coming to this stream is flushed.

The generic logical streams of any of FORT, CC, AS, or DATA and the diverted streams FORT:2, CC:2, AS:2, and DATA:2, can also be made physical by giving the file-name in the corresponding positional parameter of the program call statement for Nypatchy. Unless instructed otherwise with a +ASM line in the cradle, T=ATTACH is then assumed for such a stream.

For any other physical stream a +ASM line has to be used, for example:

```
+ASM, LATEX,   T=ATTACH   .fname      (all)
+ASM, LATEX:1, T=ATTACH   .fname      (normal)
+ASM, FORT:3,  T=ATTACH   .fname      (extra)
```

If several logical streams are to be bound to one physical stream this can be done as with this example:

```
+ASM, TEX, LATEX, T=ATTACH   .fname
```

This sends all logical streams of LATEX and TEX together through the physical stream TEX (or TEX:0, which is the same thing) onto the single file fname.

7.4) **Binding of logical streams**

"Binding" of a given logical stream means that the output of this stream goes to the same file as the output from the stream it is bound to.

Default binding is initialized such that the wanted set-up can be specified with minimal instructions. This default is:

a) the generic logical streams `CC, AS, SHELL, CRAD, DATA` are bound to the generic stream `FORT`;

b) the generic logical stream `INCL` is bound to the generic stream `CC`;

c) the generic streams of all user data types are bound to the generic stream `DATA`;

d) all logical streams `X:i` of type `X` are bound to the generic stream `X:0`; if this is itself bound to the generic stream `Y:0`, then the logical streams `X:i` are bound to the corresponding streams `Y:i`;

e) if the logical streams `X:0` or `X:1` are made physical all logical streams `X:i` are bound to it, provided they have not been bound explicitly to somewhere else; similarly if the logical stream `X:3` is made physical, the logical stream `X:4` is bound to `X:3`.

Normally re-binding by the user is done implicitely or explicitely when establishing the physical streams, for example:

```
+ASM, FORT,   T=ATTACH  .finorm
+ASM, FORT:2, T=ATTACH  .fidiv
```

collects the output of the logical streams "divert" of all data types and sends it to file `fidiv`; the output from all other streams goes to file `finorm`.

```
+ASM, DATA,   T=BYPASS.
+ASM, FORT,   T=ATTACH  .finorm
+ASM, FORT:2, T=ATTACH  .fidiv
```

does the same as the previous example, but all non-compilable material is by-passed.

Explicite binding can be done by giving several logical streams, for example:

```
+ASM, TEX:2, LATEX:2, T=ATTACH  .fname
```

which establishes the physical stream `TEX:2` and binds the logical stream `LATEX:2` to it.

To specify binding without at the same time establishing a physical stream one has to give the parameter `T=BIND`; thus to achieve the binding-only of the previous example:

```
+ASM, TEX:2, LATEX:2, T=BIND.
```

By giving 2 (or more) generic logical streams one can map all logical streams of one data type onto the logical streams of another data type, for example:

```
+ASM, TEX, LATEX, T=BIND.
```

binds the data type `LATEX` to the data type `TEX`, such that the logical streams `LATEX:i` are bound to `TEX:i`.

"Aliasing" is similar to binding, for example

```
+ASM, TEX, LATEX, T=ALIAS.
```

signals that `LATEX` is just another name for `TEX`. Only generic stream names may be aliased. The difference between binding and aliasing is further explained below and in sect. 10.

Again, several names may be given, as for example:

```
+ASM, FORT, FORT77, FORT99, T=ALIAS.
```

which tells Nypatchy that `FORT77` and `FORT99` are just other names for the data type `FORT`.

7.5) **Deck splitting with `T=SPLIT`**

One may request that Nypatchy writes each deck coming to a given physical stream to a separate file, whose name is derived from the deck-name with an extension added. For example:

```
+ASM, FORT, T=SPLIT.      (or T=MODIFY, see later)
```

requests splitting of all decks arriving on the physical stream `FORT`, that is the decks of all logical streams bound to this physical stream. It will create in the current working directory a number of files, one for each non-empty deck, with extensions like .f, or .c, or .s, or .h, added according to the data type of each deck.

Nypatchy knows the extensions needed for the standard data types `FORT`, `CC`, `INCL`, and `AS`, on the machine one is running on. For data type `CRAD` the extension used is .cra; for `SHELL` .sh is used on UNIX, .com on VAX. For data type `DATA` .dat is used on the VAX, and nothing on UNIX.

One may want to change the extensions used, or for the user data types or for type `DATA` one may want to set the extension; this can be done with, for example:

```
+ASM, FORT, T=EXT   ..ftn
```

The directory into which the files are written is the current working directory by default; this can be changed by adding a common prefix to all file names, with for example:

```
+ASM, FORT, T=SPLIT, PREFIX   .work/
```

One may request that a record of the files created be written to a file to be digested by Nyshell, a child of Fcasplit, with for example:

```
+ASM, FORT, T=SPLIT, LOG   .fname
```

The last two examples can be combined if the prefix is a pure directory, for example:

```
+ASM, FORT, T=SPLIT, PREFIX, LOG   .work/fname
```

The resulting log file will contain one line for each deck written, giving its logical stream name and its file name, as for example:

```
fort:1    chdirf.f
cc:1      chdiri.c
fort:1    getenvf.f
cc:1      geteni.c
cc:1      getpidf.c
fort:2    getwdf.f
cc:2      getwdi.c
```

Note the difference between binding and aliasing in this context: if there is a deck `M432` of type `LATEX` on the Pam file, which is processed with

```
+ASM, TEX, LATEX, T=SPLIT, LOG  ...
```

there will be an entry `"latex:1 m432"` on the log file, but if the processing is done with

```
+ASM, TEX, T=SPLIT, LOG  ...
+ASM, TEX, LATEX, T=ALIAS.
```

the entry will be `"tex:1 m432"`.

7.6) Deck splitting with `T=MODIFY`

Giving `T=MODIFY` instead of `T=SPLIT` will cause Nypatchy to compare the text of each deck against the corresponding file, if it pre-exists, and to suppress the output if the two are identical. Thus the date of the existing and unchanged file is not disturbed.

As for `T=SPLIT`, one may give, for example:

```
+ASM, FORT, T=MODIFY, PREFIX         .work/

+ASM, FORT, T=MODIFY, LOG            .fname

+ASM, FORT, T=MODIFY, PREFIX, LOG    .work/fname
```

The resulting log file will contain one line for each deck, unchanged decks are marked with the "same" flag, giving something like:

```
fort:1    chdirf.f
cc:1      chdiri.c
fort:1    getenvf.f  same
cc:1      geteni.c   same
cc:1      getpidf.c
fort:2    getwdf.f
cc:2      getwdi.c   same
```

7.7) **Routine headers**

Arbitrary routine header lines can be defined for a given data type either for all logical streams, or for a particular stream with, for example:

```
    +ASM, AS,   T=RH   .@|DECK  ID>, @ .      (all)
 or +ASM, AS:1, T=RH   .@|DECK  ID>, @ .      (normal only)
 or +ASM, AS:2, T=RH   .@|DECK  ID>, @ .      (diverted only)
```

In the comment field of the control line one gives the mask for generating the routine header line; the very first character is the escape character used to indicate the place where the routine name has to be inserted.

On the machines where this useful, routine header lines are pre-initialized as needed by Fcasplit on the machine where one is running on to something like:

```
+ASM, FORT, T=RH   .@CDECK  ID>, @ .
+ASM, CC,   T=RH   .@/*DECK ID>, @ . */
+ASM, AS,   T=RH   .@|DECK  ID>, @ .
```

This can be switched off, for example with:

```
+ASM, AS, T=RH.
```

with an empty comment field.

Routine header lines are not generated for processing in `T=SPLIT` mode.

Note that `+ASM, T=RH...` lines merely modify the information for the logical stream, quite independent from and without disturbing its binding to some physical stream.

7.8) **C/C substitution**

In a similiar manner, control-character substitution can be requested with, for example:

```
+ASM, DATA, T=CCHSUB   .&+!+
```

This example changes the control-character `&` or `!` on any line of the output written to the ASM file for all logical streams of type `DATA` into a `+` if the key of the line is that of a Patchy control line.

The data types `SHELL` and `CRAD` are initialized with `T=CCHSUB .&+` which can be over-ridden with a `+ASM` line. Again, this could be switched off with `+ASM, SHELL, T=CCHSUB.` for example.

7.9) **Correcting a data type**

Primarily to help in the phase of transition from Patchy 4 to 5, a facility is provided to change the data type of all decks of a particular logical stream, for example:

```
+ASM, DATA:2, T=REASSIGN, S=INCL.
```

This tells Patchy that material driven to the logical stream `DATA:2`, "diverted DATA", consists really of include files for C. This material should have had a `T=INCL` declaration on the Pam file and it will be treated as if it had: it will be sent to output through the logical stream `INCL:2`, with the right properties, like file extension or routine-header lines.

This also works:

```
+ASM, CC:2, T=REASSIGN, S=INCL.
```

"Diverted CC" will be re-assigned to "diverted INCL", which logical stream is bound be default to `CC:2`, and if this latter is made physical the output will appear there, but with the properties corrected.

Note that "reassignment" has nothing to do with "binding" and does not interfer with it.

7.10) User type confirmation

All user data types whose material is USEd should have been declared on some `+ASM` line in the blank deck of the cradle.

If no `+ASM` line for such a user data type is needed for any of the above purposes, a line like

```
+ASM, LATEX, T=USED.
```

should be given, for example; it does nothing but open the generic logical stream `LATEX`.

Several data types may be given on one line as with

```
+ASM, LATEX, TEX, SCRIPT, T=USED.
```

which does not imply any binding.

If Nypatchy finds a non-declared data type when processing USE-selected material on the Pam file, it prints a warning and opens a generic stream for this data type, which it then binds to the stream `DATA`.

7.11) Summary of control lines

The general format is:

```
+ASM, (D=)str1,str2,..., T=action, S=dtyp, IF=...   .comment-field
```

Establishing a physical stream, with binding:

```
+ASM, phys, log1, log2, ..., T=ATTACH   .fname
+ASM, phys, log1, log2, ..., T=SPLIT.
+ASM, phys, log1, log2, ..., T=MODIFY.
+ASM, phys, log1, log2, ..., T=BYPASS.
```

Binding only:

```
+ASM, here, log1, log2, ..., T=BIND.
```

Aliasing:

```
+ASM, here, generic1, generic2, ..., T=ALIAS.
```

Specify file-name prefix for `SPLIT` or `MODIFY` mode:

```
+ASM, phys, T=PREFIX   .prefix
```

Specify log file for `SPLIT` or `MODIFY` mode:

```
+ASM, phys, T=LOG      .fname
```

Specify default extension:

```
+ASM, log1, log2, ..., T=EXT.   extension
```

Specify routine header lines

```
+ASM, log1, log2, ..., T=RH  .mask
```

Specify control character substitution:

```
+ASM, log1, log2, ..., T=CCHSUB  .mask
```

Data-type reassignment

```
+ASM, log, T=REASSIGN, S=true.
```

User type declaration:

```
+ASM, log1, log2, ..., T=USED.
```

7.12) Back-compatibility processing

To allow a period of transition where one may want to use the same cradles both with versions 4 and 5 of Patchy, the following is available in back-compatible mode (the default for the time being):

If Nypatchy is presented with a `+ASM` line for the old Ypatchy it will do the following:

```
a) translate  21 -> FORT:1,  22 -> AS:1,  23 -> DATA:1,  24 -> CC:1
              31    FORT:2,  32    AS:2,  33    DATA:2,  34    CC:2

b) a parameter LUN=n is ignored.

c) if no T= parameter is given T=BIND is assumed.

d) the presence of a parameter like R=! will cause an error.

e) the presence of any of T=EOF, HOLD, MIX, RESUME will cause
   an error (rather unlikely to happen).
```

Thus the following cradle is alright as it stands, provided one does not want to use the new features of Patchy, such as Nyshell:

```
+USE, ${PLINAME}.
+ASM, 23, T=ATT      .=${CERN_ROOT}/bin/wylbur.help
+ASM, 24.
+EXE.
+USE, *WYLBUR.
+PAM, 11, T=C,ATT    .${CERN_ROOT}/src/car/wylbur
+QUIT.
```

The line `+ASM,23,T=ATT ...` is translated into `+ASM,DATA,T=ATTACH ...` The line `+ASM,24.` is translated to `+ASM,CC,T=BIND.` and has hence no effect: with Patchy 4 this line establishes stream 24 with the file-name taken from the command line. With Patchy 5 the presence of the command line parameter for stream CC is sufficient; but be careful:

the order of the file-names on the program-call statement has changed:

```
    asm=     21                 22 23   31     32     33     24   34
 ypatchy pam fort cradle print as data fort:2 as:2   data:2 cc   cc:2

nypatchy pam fort cradle print cc as   data   fort:2 cc:2   as:2 data:2
    asm=     21                 24 22   23     31     34     32   33
      p= 1   2    3      4      5  6    7      8      9      10   11
```

Remember also the default stream binding rules:

With Patchy 4 all streams are initially bound to stream 21; to establish a physical stream "n" a control-line `+ASM,n` has to be given as well as the file-name in the correct position on the program-call statement.

With Patchy 5 generic streams

```
CC, AS, SHELL, CRAD, DATA  are intially bound to stream FORT,
stream  INCL is initially bound to CC,
all user data types are bound to DATA,
logical streams X:i are initially bound to X:0, but if X:0
is itself bound to Y:0 all X:i are bound to Y:i.

Giving a file-name for a particular stream on the program-call
command establishes this stream as physical.
```

For the following Cernlib cradle:

```
+USE, ${PLINAME}.
+ASM, 23, T=ATT    .:flibD.dat
+ASM, 24.
+ASM, 34, R=!      ./*DECK ID>, !.h */
+EXE.
+DIVERT, FMH.
+USE, *FATLIB.
```

```
...
+PAM, 11, T=C,ATT     .${CERN_ROOT}/src/car/fatmen
+QUIT.

being used with:
              21=FORT:1           22    24=CC:1   34=CC:2
            1 2         3     4   5     10        11
    ypatchy - flibf.fca $CRAD $PR ----- flibc.fca flibc.inc .go
    fcasplit flibf.fca
    fcasplit flibc.fca
    fcasplit flibc.inc
```

we have to do something; let us change it into:

```
+USE, ${PLINAME}.
+ASM, 23, T=ATT    .:flibD.dat
+ASM, 24.
+ASM, 34, R=!,  IF=-PY_VS5     ./*DECK ID>, !.h */
+ASM, 34, T=REASSIGN, S=INCL, IF=PY_VS5.
+EXE.
+DIVERT, FMH.
+USE, *FATLIB.
...
+PAM, 11, T=C,ATT     .${CERN_ROOT}/src/car/fatmen
+QUIT.
```

This cradle can be used with Patchy 4 as before, provided that `P=PY_VS5` is not USE selected.

It can be used with Patchy 5 in the same context:

```
               21=FORT:1           24=CC:1   22   34=CC:2   32
             1 2         3     4   5         6    9         10
    nypatchy - flibf.fca $CRAD $PR flibc.fca ---  flibc.inc .go
    fcasplit flibf.fca
    fcasplit flibc.fca
    fcasplit flibc.inc
```

`P=PY_VS5` is internally pre-selected as USEd if Patchy 5 is running.

8 Patchy 5 on the IBM with VM/CMS

On the CERN IBM all programs except Nyshell are available; they are supposed to behave as under UNIX or VAX VMS. Thus they are modules ready to execute without an EXEC interface, capable to dialog with the user if necessary, for example:

```
nypatchy  zebra zebra zeblib .go

   will read the Pam file "ZEBRA CAR A",
       write the ASM file "ZEBRA FORTRAN A",
       using the cradle   "ZEBLIB CRA A",
   and the printed output will go to "standard output".
```

If the list of files on the command line is not declared to be complete by giving ".go", the Patchy modules will open a dialog with the user prompting him for the file names (or options).

File names on the command line, or on control-lines `+PAM` or `+ASM` with the `T=ATTACH` option, are given in Unix style, for example:

```
zebra.fortran.a  or  a/zebra.fortran  to mean "ZEBRA FORTRAN A",

thus    nypatchy  x/zebra b/zebra.fca b/zeblib  .go

        would use files  "ZEBRA CAR X", "ZEBRA FCA B", "ZEBLIB CRA B".
```

If the file-mode (equivalent to the Unix directory) is not specified file-mode A is used by default.

For more details about the dialog and syntax one may type for example `nylist help`. To get help about the auxiliary programs one can give the H option, for example `nylist - h`. Note that to open an output file for APPEND one has to prefix the file name with +, for example:

```
nyindex  zebra -  ze  .go
nylist   zebra - +ze  .go
```

For input files with file-type CAR or CRA which are not found, the file-type CARDS or CRADLE is tried for back-compatibility.

Because the Fortran compiler collapses if it is presented with a file for compilation containing lines longer than 80 characters, nypatchy will truncate such lines on output for decks whose data-type is `FORT` (IBM only).

9 The new Patchy auxiliary Nydiff

Nydiff compares two different evolutionary stages of the same Pam file and delivers the differences found as a cradle which can be used to update the earlier stage to the later stage. This allows to collate the updates from parallel developments started from the same original Pam file.

Beyond the normal operations of changing the contents of decks, of deleting patches or decks, and of adding new patches or decks, the user is supposed to be also free to some extent to change the names of patches and decks, and to re-arrange the order of the patches on their Pam file, or of the decks within their patch.

Although it would always be possible to make a cradle to update even two unrelated files, to be useful when comparing several parallel evolutions of the same Pam file it is essential to minimize the volume of the cradle, which means that Nydiff has to do its best to determine which deck on "old" is at the origin of which deck on "new".

To do this Nydiff operates in the following stages:

1) the number of Pam files must be the same on "old" and "new", their order is assumed not to have changed, but the Pam file identifiers may have changed; migration of material from one Pam file to another one is assumed not to occur.

2) matching by name assumes that patches of the same name on "old" and "new" are related, and also decks of the same name in related patches. At the end of this process there is a test (which can be bypassed with the A option): unless 3/4 of the patches have been matched by name the error exit is taken.

3) matching of patches by content compares all as-yet unmatched patches on "old" against all unmatched patches on "new" (of the same Pam file): if either 3/4 of the deck names or 80 per-cent of the content are the same the two matching patches are assumed to be related.

4) matching of decks by content compares all as-yet unmatched decks of each patch on "old" against all unmatched decks in the related patch on "new": if the contents of two decks are the same to 80 per-cent they are supposed to be related. At the end of this process there is a second test (which can be by-passed with the F option): unless 3/4 of all decks have been matched the error exit is taken.

5) to find "lost" decks which may have been tranfered from one patch to an other one, Nydiff takes each unmatched deck on "old" (of a non-deleted patch) and scans the patches on "new" to compare unmatched decks by name and by content: if the name is the same and the content the same to the 80 per-cent level the deck is pulled from its "old" patch.

On the result of the matching process the cradle is built: patches or decks of "old" not found on "new" are deleted with `+USE, T=INH`, patches or decks on "new" not found on "old" are added with `+ADD`. For related decks the differences of content are recorded with `+ADD`, `+REPLACE`, or `+DELETE`.

Since re-ordering the material of a Pam file cannot be done with Nypatchy if one also wants to have a minimal cradle, the instructions to re-order are delivered as an Exec-file for the Wylbur editor.

The program call for Nydiff is:

```
nydiff  old.car  new.car  diff.ucra  options  print

        old  is the reference Pam file from which
        new  has been derived;

  diff.ucra  is the cradle for Nypatchy delivered to update the
             decks on "old" to become "new",

  diff.uexe  is the Exec file for Wylbur to re-order the file;

   options:  A  (anyway) by-pass the intial test to see whether
                3/4 of the patches have been identified between
                "old" and "new",

             F  (force) by-pass also the later test to see whether
                3/4 of the decks have been identified by name or
                by content between "old" and "new"

      print  is the printed output, stdout by default.
```

If one has the reference Pam file `old.car`, say, and one receives the result from Nydiff in the form of the two cradles `diff.ucra` and `diff.uexe`, one can make `new.car` with this shell script:

```
nypatchy  -  new.car  .go  <</
+UPDATE.
+LIST, UPD.  if one wants to see the corrections in context
+USE, T=EXE.
+PAM, T=ATTACH       .diff.ucra
+PAM, T=ATT, UPDATE  .old.car
+QUIT.
/

useb  new.car  <</
@diff.uexe
save,unn,replace
quit
/
```

The correction cradle diff.ucra comes in fact in the form of a Pam file; all corrections are headed with `+PATCH,UPD.`

"useb" calls a version of Wylbur which can handle a file of up to 120 000 lines; the normal version of Wylbur, called with "use" can handle 60000 lines.

Example

Suppose two people in Oslo and Lund each start a development of the same file zebra.car and end up with 2 new Pam files zoslo.car and zlund.car. To check the compatibility of their changes (by eye) one can do this:

a) extract the correction cradles:

```
nydiff zebra.car zoslo.car oslo.ucra .go
nydiff zebra.car zlund.car lund.ucra .go
```

b) change the patch name of the corrections to identify the origin:

```
use oslo.ucra   <</
change 'UPD' to 'OSLO' in '+PAT'1 xqt 1
sur
use lund.ucra
change 'UPD' to 'LUND' in '+PAT'1 xqt 1
sur
/
```

c) make a collated listing of all changes:

```
nypatchy --- change.lis .go  <</
+UPDATE.
+USE.
+LIST, OSLO, LUND.
+PAM, T=ATT      .oslo.ucra
+PAM, T=ATT      .lund.ucra
+PAM, T=ATT,UPD  .zebra.car
/
```

10 The new Patchy auxiliary Nyshell

Nyshell used in conjunction with Nypatchy is intended to replace the use of Fcasplit of Patchy version 4, and that part of the "make" utility which has to do with the re-compilation of modified routines.

Remember: Ypatchy of version 4 writes the EXE selected decks together to an ASM file. Fcasplit reads this file, splitting the routines into separate files, writing out a shell-script (or a make file) to compile all the routines.

With Patchy version 5, the splitting onto separate files is done already by Nypatchy, which writes on a log file the names of all routines written, together with the logical stream name of each routine. Using this log Nyshell will create a shell-script to compile the routines which need compilation.

Moreover, if the splitting has been done to a physical stream established with `+ASM,...,T=MODIFY,LOG` the re-compilation of unchanged routines can be avoided. But this implies that at least the .f files are retained indefinitely (the .o files could be restored from the library).

10.1) Deck splitting having been done with `T=SPLIT`

In this mode the use of Nyshell is straight-forward, Nyshell simply converts the log file into a shell-script to compile all routines.

10.2) Deck splitting having been done with `T=MODIFY`

The log file from Nypatchy will contain one line for each routine, giving the stream name and the file name. Unchanged routines have additional information: the "same" flag plus the names of the files called with `+INCLUDE`, if any; something like:

```
fort:1   chdirf.f
cc:1     chdiri.c
fort:1   getenvf.f  same
cc:1     geteni.c   same
cc:1     getpidf.c
fort:1   getwdf.f
cc:1     getwdi.c   same
cc:1     fzincc.c   same   fzinc.h mzinc.h
```

(This last line indicates that on stream cc:1 the source file fzincc.c is unchanged, and that it calls the include files fzinc.h and mzinc.h, also unchanged, and also residing in the same source directory as everybody else.)

In `MODIFY` mode we can avoid re-compiling routines which have not changed. But we need more: routine names may have changed, or some routines may have been removed from the Pam file.

Also, the previous run may have abended, such that the new .f files have not yet been compiled, but the second Nypatchy run will see unchanged .f files.

Nyshell has been programmed to handle the following frequent situation: the current working directory contains all and only the .o files of all the routines of the program (or the package) being dealt with. When the compilation is done, all .o files will be collected into one .a library file.

In this situation one gives the U (uptodate) option to Nyshell which will then do the following:

a) ready for compilation all routines without the "same" flag;
b) ready for compilation all routines appearing in the log file for which the .o file is not more recent than the .f (or .c, .s) file, and not more recent than all the `INCLUDE` files called, if any;
c) ready for compilation all routines for which the compiler options have changed since the last run (see para.4);
d) check all .o files in the current directory against the log file: delete any file if its name does not appear there, or if it belongs to a routine which must be re-compiled.

On the VAX `.f` is `.FOR`, `.s` is `.MAR`, `.o` is `.OBJ` and `.a` is `.OLB`.

10.3) Controlling the compilations

Nyshell has a default set-up built into it on how to compile normally on the machine it is running on. The default set-up can be seen by typing

```
nyshell - h
```

This default set-up can be changed by control-lines given in the cradle. The cradle may be given as a here-document read from standard input; it may be declared to be empty by giving `EOF` as file name; or it may be read from a file whose name is given on the command line in the "read" position. If, with this last case, the file does not exist, this does not create an error but is accepted as a signal to use the default set-up as is.

As an example, here are the cradle lines which would produce the default set-up on the Apollo:

```
fo:1     -opt 3
fo:2     -opt 0
fo:3     =:1
fo:4     =:2
fopt     -bounds_violation -info 1 -indexl -cpu mathlib_sr10
fc       /com/ftn
fort:1   ${FC} ${s}$*.ftn ${FO_1}; mv $*.bin $*.o
fort:2   ${FC} ${s}$*.ftn ${FO_2}; mv $*.bin $*.o
fort:3   =:1
fort:4   =:2

co:1     -O
co:2
co:3     =:1
co:4     =:2
copt     -c
cc       cc
cc:1     ${CC} ${CO_1} ${s}$*.c
cc:2     ${CC} ${CO_2} ${s}$*.c
cc:3     =:1
cc:4     =:2

ao:1
ao:2     =:1
ao:3     =:1
ao:4     =:2
aopt
```

```
as       as
as:1     ${AS} ${AO_1} ${s}$*.s
as:2     =:1
as:3     =:1
as:4     =:2

start    # Script from Nyshell for file f.e
end      # End of the shell script
```

The tag on the left identifies the information given for it on the right. Thus "fc", "cc", and "as", specify the command verbs to be used to call the compiler for Fortran, C, and assembler, respectively.

"fopt" gives the compiler options to be used for all logical streams; whilst "fo:i" gives the extra options to be used for stream `FORT:i`. "fort:i" specifies the complete command to be given to the shell to compile one routine, the escape `$*` indicating the place(s) where the routine name has to be inserted. The value `"=:1"` means: same as stream 1.

Tag "sdir" may be used to explicitly set the directory where the source files are; by default this is the directory where the log file is found.

The default set-up assumes that routines coming to streams :1 and :3 are to be compiled normally, and that routines coming to streams :2 and :4 have problems and should be compiled without optimisation.

One may over-rule the default with a line like:

```
fopt     -bounds_violation -info 1 -indexl -cpu mathlib_sr10 -dbs
```

or one may add to the default with a line like:

```
+fopt     -dbs
```

From the set-up Nyshell constructs shell parameter definitions followed by the shell commands to compile. As an example, here is the shell script delivered for the latest re-making of my debug version of KERNLIB on Apollo:

```
#!/bin/csh -f
# Script from Nyshell for file  ../source.log
set FILE = source
set s    = "../"
set FO_1 = "-bounds_violation -info 1 -indexl -cpu mathlib_sr10 -dbs -opt 3"
set FC   = "/com/ftn"
set CO_1 = "-c -g -O"
set CC   = "cc"
${CC} ${CO_1} ${s}geteni.c
${CC} ${CO_1} ${s}getgidf.c
${CC} ${CO_1} ${s}getpidf.c
${CC} ${CO_1} ${s}getuidf.c
${CC} ${CO_1} ${s}getwdi.c
${FC} ${s}gshowf.ftn ${FO_1}; mv gshowf.bin gshowf.o
${FC} ${s}ccopiv.ftn ${FO_1}; mv ccopiv.bin ccopiv.o
${FC} ${s}ccosub.ftn ${FO_1}; mv ccosub.bin ccosub.o
${FC} ${s}mxmad.ftn  ${FO_1}; mv mxmad.bin  mxmad.o
${FC} ${s}mxmlrt.ftn ${FO_1}; mv mxmlrt.bin mxmlrt.o
${FC} ${s}mxtrp.ftn  ${FO_1}; mv mxtrp.bin  mxtrp.o
${FC} ${s}mxuty.ftn  ${FO_1}; mv mxuty.bin  mxuty.o
# End of the shell script
```

resulting from this call to Nyshell:

```
nyshell ../source.log  u  .go <</
  +fopt    -dbs
  +copt    -g
/
```

On the VAX, Nyshell will itself substitute the parameter values into the compilation command lines; for this the parameters have to have the curly brackets as shown; Nyshell will look for the tokens `${s}, ${FC}, ${FO_1}`, etc.

Note that Nyshell itself and the resulting shell-script `source.shfca` must be run in the directory where the .o files are or are to be.

10.4) Detecting change in compiler options

The last step in the execution of Nyshell is to write to a file, the ".xqtlog" file, a list of all the routines involved and their properties. This contains the compiler options for each stream, and the names of all routines belonging to each stream.

Next time round, Nyshell reads the .xqtlog file from the last run to check for each routine that its compiler options are unchanged. Thus the complete set of conditions for a routine not to need re-compilation is:

```
1) Nypatchy sends the "same" flag;

2) the current compiler options are the same as on .xqtlog;

3) the .o file exists and is more recent than the .f (or .c) file,
   and more recent than any .h file used;
```

The use of the input .xqtlog file can be by-passed with the "B" option; the "E" option requests by-passing if this file is not existing. If neither B nor E is given, Nyshell will trigger recompilation of all routines if it does not find the .xqtlog file in the current directory to be valid.

The name of the .xqtlog file is derived from the name of the log file given to Nyshell; in the examples given in this paper this name would be `source.xqtlog`.

Note that Nyshell does of course not analyse the meaning of the compiler options, thus any trivial change will force recompilation.

10.5) The program call for Nyshell is:

```
nyshell  x.log  options  read  print

    "x.log":  the input log file having been created by Nypatchy;
              the output files delivered will be
              x.shfca  the shell script to be run for compilation
              x.xqtlog remembering the compilation options

  "options":  A  all - compile or re-compile all files
              B  bypass the use of the input .xqtlog file
              E  empty - bypass the .xqtlog file if it is empty
              H  help - print the help information only
              Q  quick - do not display the set-up information
              U  uptodate - check that all .o files in the
                            current directory are ready to be
                            put into the last-version library
              V  verbose - display the complete set-up.

     "read":  name of the file with the user set-up commands,
              EOF  if default set-up to be used as is,
              default: standard input

    "print":  printed output file, default: standard output
```

10.6) Example 1: my job to re-make my Patchy on Apollo

```
#!/bin/csh  -v -f
# Create Patchy 5 library and modules

  set  PRO     = ~/bin
  set  NEW     = ~/p5/dev
  set  PAM     = ~/p5/wyl/patchy.car
  set  KERNLIB = ~/kern/inst/libkerndbg.a
  set  EXTRA   = ' '

  setenv  ACTION    MODIFY
  setenv  MACHINE   APOLLO
     set  LOAD    =  ld

 if (-d wk_patchy == 0)  mkdir wk_patchy
 cd     wk_patchy

 $PRO/nypatchy  $PAM  .go  <<\\
+ASM, FORT, T=${ACTION}, LOG   .source.log
+EXE.
+USE, *PATCHY, ${MACHINE}.
+USE, QDEBUG, TESTR.
+OPT, MAP.
+PAM.
\\
 set rc = $status
 if ($rc != 0)  exit

 $PRO/nyshell source  u  .go <<\\
 +fopt   -dbs
\\
 set rc = $status
 if ($rc != 0)  exit

 csh -f -v  source.shfca

 if (-f p5lib.a)  rm p5lib.a
 ar rc  p5lib.a   *.o

 $LOAD -o $NEW/nypatchy  npatch.o   p5lib.a $KERNLIB $EXTRA
 $LOAD -o $NEW/nyindex   nindex.o   p5lib.a $KERNLIB $EXTRA
 $LOAD -o $NEW/nylist    nlist.o    p5lib.a $KERNLIB $EXTRA
 $LOAD -o $NEW/nyshell   nshell.o   p5lib.a $KERNLIB $EXTRA
```

10.7) Example 2: my job to make my KERN libraries

```
#!/bin/csh -f -v
# Shell script to create short KERN libraries on Apollo

 setenv KERNMACH   ~/kern/pam/kernapo.car
 setenv KERNFOR    ~/kern/pam/kernfor.car

 set KERNLIB   = ~/kern/inst/libkernsh.a
 set KERNLIBDB = ~/kern/inst/libkerndbg.a

 set MICKYLIB  = ~/kern/lib/libmicky.a
 set MICKYPAM  = ~/kern/pam/micky.car

 cd   ~/kern/inst
```

```
 if (-d wk_kern == 0)  mkdir wk_kern
 cd     wk_kern

 nypatchy  .go  <<\\
+ASM, FORT, T=MODIFY, LOG  .source.log
+EXE.
+USE, *KAPO.
+PAM, T=ATTACH   .$KERNMACH
+PAM, T=ATTACH   .$KERNFOR
+QUIT.
\\
 if ($status != 0)  exit

#  ----------------------------------------  compile debug library

 if (-d wk_dbg == 0)  mkdir wk_dbg
 cd     wk_dbg                          | cd is inst/wk_kern/wk_dbg
                                        |
 nyshell ../source  u  .go <<\\         |
 +fopt    -dbs                          | take the source from ../
 +copt    -g                            | and re-compile the changes
\\                                      | into ./
 if ($status != 0)  exit

 csh -f -v  source.shfca

 if (-f $KERNLIBDB)  rm $KERNLIBDB
 ar rc  $KERNLIBDB  *.o
 cd ../

#  ----------------------------------------  compile normal library

 nyshell source  uq  eof  .go           |
                                        | cd is  inst/wk_kern
 if ($status != 0)  exit                | take the source from here
                                        | and re-compile the changes
 csh -f -v  source.shfca                | into here
                                        |
 if (-f $KERNLIB)  rm $KERNLIB
 ar rc  $KERNLIB  *.o

 cd ../

#  ----------------------------------------  Run Micky test job
 nypatchy  $MICKYPAM  mickyrun  .go  <<\\
+EXE, MICKY, D=MIMAIN.
+USE, *MAPO.
+PAM.
\\
 /com/ftn mickyrun -bounds_violation -info 1 -indexl -cpu mathlib_sr10
 ld -o mickyrun mickyrun.bin $MICKYLIB $KERNLIB
 mickyrun
```

10.8) Example 3: a job to make the ZEBRA libraries

Here we use `+XDIVERT` to make two streams into 2 separate subdirectories.

```
#!/bin/csh -f
```

```
# Shell script to create the ZEBRA libraries

 set PAM    = ~/zebra/pro/zebraq.car
 set ZLIB   = ~/zebra/pro/zebra.a
 set ZLIBTE = ~/zebra/pro/zebrate.a

 cd   ~/zebra/pro

 if (-d wk_lib  == 0)  mkdir wk_lib
 if (-d wk_test == 0)  mkdir wk_test

 nypatchy  $PAM   .go  <<\\
+ASM, FORT,   T=MODIFY, PREFIX, LOG  .wk_lib/source.log
+ASM, FORT:3, T=MODIFY, PREFIX, LOG  .wk_test/source.log
+EXE.
+USE, QTESTLIB, T=XDIV, TRANS.
+USE, *ZEBRAL.
+USE, FZCONV, T=INH.
+OPTION, ALL, MAP.
+PAM.
+QUIT.
\\
 if ($status != 0)   exit

#   ------------------------------     Compile for library zebra.a

 cd    wk_lib
 if (-f $ZLIB)  ar xo $ZLIB         | restore the .o files

 nyshell source  u  eof   .go

 set rc = $status                   |
 if ($rc == 1)   goto libok         | status 1: no new compilations
 if ($rc != 0)   exit               |        2: error

 csh -f -v  source.shfca

 if (-f $ZLIB)  rm $ZLIB
 ar rc  $ZLIB  *.o
libok:
 touch  $ZLIB
 rm  *.o                            | delete the .o files

#   ------------------------------     Compile for library zebrate.a

 cd ../wk_test

 nyshell source  u  eof   .go

 if ($status != 0)   exit

 csh -f -v  source.shfca

 if (-f $ZLIBTE)  rm $ZLIBTE
 ar rc  $ZLIBTE  *.o
```

In this example we delete the .o files and restore them on the next run from the library file; this saves disk space but costs time.

10.9) Example 4: my job to re-make my Patchy on the VAXes

With these commands in the login file `login.com` to switch between VAX and Alpha:

```
$ if f$getsyi("ARCH_NAME").eqs."VAX"
$ then  DOING :== VAX
$       assign disk$support:[zoll.vaxnew]   NEW
$       assign disk$support:[zoll.vaxexe]   EXE
$       assign disk$support:[zoll.vaxlib]   LIB
$ else  DOING :== ALPHA
$       assign disk$support:[zoll.alphanew]   NEW
$       assign disk$support:[zoll.alphaexe]   EXE
$       assign disk$support:[zoll.alphalib]   LIB
$ endif
```

we can run this:

```
$ set verify = procedure
$ set verify = noimage
$ on error     then $ goto exit
$ on control_y then $ goto exit
$!
$!   COM-file to create nypatchy.EXE etc. using Patchy 5 for VAX or ALPHA
$!
$ assign  [zoll.p5.dev]   OPDIR
$ assign  [zoll.p5.wyl]   SRCDIR
$ assign  lib:kernlib     KERNLIB
$ assign  lib:p5lib       P5LIB
$ ACTION :== MODIFY
$!
$ set default OPDIR
$ set default [.wk_patchy]
$ nypatchy  SRCDIR:patchy.car  .go
+ASM, FORT, T=${ACTION}, LOG   .source.log
+EXE.
+USE, *PATCHY, VAX.
+OPTION, MAPASM.
+PAM.
+QUIT.
$ set default [.wk_'DOING]
$ nyshell ../source.log u eof  .go
$ if $status .ne. 1 then goto exit
$ @source.shfca
$ purge/nolog
$ lib/create P5LIB *.obj
$ ULIB  :== KERNLIB/LIB,SYS$LIBRARY:VAXCRTL/LIB
$ link/nomap/exe=NEW:nypatchy  P5LIB/INC=npatch/LIB,'ULIB
$ link/nomap/exe=NEW:nyindex   P5LIB/INC=nindex/LIB,'ULIB
$ link/nomap/exe=NEW:nylist    P5LIB/INC=nlist/LIB,'ULIB
$ link/nomap/exe=NEW:nyshell   P5LIB/INC=nshell/LIB,'ULIB
$exit:
$ set default OPDIR
$ set noverify
$ exit
```