Kernel Developers' Manual

Michael W. Young
Mary R. Thompson

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

Version of:

10 August 1989

Abstract

This document describes the organization of the official kernel sources and build areas, the recommended methods for getting a private copy of selected kernel sources and building your own version of the kernel, and the required way of inserting your changes back into the official sources. It is intended to be used by the kernel developers at CMU.

1. Introduction

This document describes the organization of the Mach kernel sources and the recommended ways of makeing changes to them. This document is mainly useful to kernel maintainers at CMU, but it includes some general information about choosing kernel configurations that might be useful to kernel maintainers outside of CMU.

2. Background knowledge

2.1. RCS

RCS is used extensively to maintain a history of modifications to the kernel sources, and to aid in managing concurrent lines of development. Therefore, a working knowledge of RCS is desirable. See the online manual pages for *rcs* for a discussion of how revisions are managed. Most of the local changes to the RCS programs were done to make the version control of a set of files stored in a directory tree easier. Configuation rules and snapshot files have been introduced to facilitate the specification of which version to chose from a set of files. Snapshot files are used to specify the version of each file that make up a complete version of a set of files. Configuration rules specify some general rule like a date or branch name which is used to select the desired version of each file.

A set of *bscripts* have been written to keep track of all the files and changes on a branch and to handle the problems that arise from the separation of the RCS tree from the checked out versions of the files. Further information can be found in the RCS cookbook which can be found in /afs/cs/project/mach/doc/rcs-cookbook.PS and the man page for bci(1).

2.2. Andrew File System

The Andrew File System (AFS) is used to hold all of the kernel sources and compiled object files. An understanding of the AFS filesystem in general, and its protection scheme in particular, are required. It should be noted that all the processes on a machine with the same userid share the same AFS authentication token. Thus if you run several processes under the same userid and one of them changes the AFS authentication, the authentication changes for all the processes with that userid.

AFS recognizes the special name *@sys* in symbolic link names. When evaluating such a link name *@sys* is bound to *vax_mach, ibmrt_mach* or *sun3_mach* depending on what type of machine the task is executing on.

2.3. Make

The local version of *make* has been modified recently to deal with the problem of building programs for serveral machine types from one set of sources. Features have been added to allow the objects to be built in a directory other than the source directory. The *make* variable *OBJECTDIR* is used to specify where the object files should be placed. The built-in *make* variable *machine* is set to *vax, ibmrt* or *sun* depending on what machine type the *make* is being executed on. *OBJECTDIR* is typically defined in a file called *Makeconf* as a function of *machine* and relative to the source tree. *make* will search up the tree starting from the current working directory looking for a *Makeconf* file. The *-N* switch to *make* will supress this search.

2.4. SUP

Sup is a program used at CMU to copy a consistent set of files from one machine to another. In our current source distribution *sup* is only used to provide compatibililty for those machines that do not have access to AFS. A copy of the latest and alpha sources is kept on wb1.mach.cs.cmu.edu as a *sup* repository. For information on how to use *sup* see the manual page sup(1).

2.5. Other

Note that the machine names used by AFS: (*vax_mach, ibmrt_mach, sun3_mach*, those used by *make*: *vax, ibmrt, sun*, and those used by the kernel: *vax, ca, sun3* are all slightly different. This difference accounts for the indirect linking to machine directories, and the various names for machine dependent stuff.

The kernel specific scripts that are mentioned in this document can be found in */usr/mach/etc* or */afs/cs/mach/src/kernel/latest/src*. The best versions of the rcs programs and bscripts can be found in */afs/cs/@sys/{alpha,beta,omega}/usr/misc/.rcs*. To get these on your machine use

```
modmisc -release beta cs.misc.rcs
```

If you set *-release alpha* you will get any new features more quickly. *build* and *workon* are released as another misc collection: cs.misc.sdm. These program can be linked to in */afs/cs/@sys/{alpha,beta,omega}/usr/misc/.sdm* or supped to your machine by using the following command.

```
modmisc cs.misc.sdm
```

3. Organization of kernel source, release and build trees

The kernel source tree is defined by the mainline revision from the RCS tree rooted at */afs/cs.cmu.edu/mach/rcs/kernel*. This tree is structured exactly as the checked-out tree, and contains only files of the form *foo,v*. For example, the source file *conf/Makefile.template* lives in the RCS tree as *conf/Makefile.template,v*.

The mainline revision is left checked-out (unlocked, meaning that they are read-only) in a tree rooted at */afs/cs.cmu.edu/mach/src/kernel/latest*. This area is guarenteed to be in a consistant state only if the *.BCSlock* file in that directory is empty. The checked out tree does *not* include embedded *RCS* directories or links to them. This differs from how most installations use RCS. This division has the advantage of providing a clean set of sources to be used by *find* or *grep*. The *bscripts* versions of RCS commands can deal with this separation.

Occasionally, the latest kernel sources will be moved into the **alpha** release, and kernel binaries will be distributed to appropriate machines. The sources corresponding to various releases are left checked-out in */afs/cs.cmu.edu/unix/source/{alpha,beta,omega}/kernel*.

Directories containing kernel compilations for various configurations are kept rooted at */afs/cs.cmu.edu/mach/obj/@sys/kernel/latest*. Binaries for released versions can be located by replacing *latest* with the appropriate release name.

Each time the mainline is updated a variety of useful information about the release is created:

- The file *conf/version.edit* is incremented by one and checked into the RCS tree.

- The file *../kernel/latest/.BCSset-TRUNK* which is a list of all the files that have changed between this version and the previous one is created and checked in to the RCS tree.
- A set of complete snapshot files is generated and placed in *.../kernel/latest/SNAPSHOTS* and the RCS tree. These snapshot files are named *ALL, MI, ca, vax, sun3, mmax, pmax ,i386*, and *sqt* and correspond to all the files in the kernel, all the machine independent files, and all the files for each machine type.
- The entire tree is supped to */../wb1/usr/kernel/latest*.
- A post is made to the mach-kernellog bboard with a summary of all the changes to this version of the kernel.
- The new version number and date is placed in *.../kernel/latest/VERSIONS*.

4. Kernel AFS groups and privileges

To get write permission to any of the official kernel areas on AFS you must authenticate to AFS as one of three users: *kernel, kernsrc* or *kernbin*. All three users have all Kernel viewing privileges, i.e are members of Kernel:View.{IBM,SUN,ULTRIX,4BSD,SEQUENT,ENCORE,Local,NFS,Any} and have all System viewing privileges, i.e. are members of System:View.{IBM,SUN,ULTRIX,4BSD,SEQUENT,ENCORE,Local,NFS,Any}.

The write permissions of the three users are:

kernel — is a member of Kernel:UpdateArchive which allows modification of the /afs/cs/mach/rcs/kernel subtree.

kernsrc — is a member of Kernel:UpdateSource which allows modification of the /afs/cs/mach/src/kernel/{latest,alpha,beta,omega} subtrees.

kernbin — is a member of Kernel:UpdateBinary which allows modification of the /afs/cs/mach/obj/@sys/kernel/{latest,alpha,beta,omega} subtrees.

Before you authenticate to AFS as a special user, you should *su* or *nu* to set your userid to that user. This will prevent you from inadvertly changing the AFS authentication of other processes using your personal userid.

If you are going to check out files from the kernel sources using *machtree* or an update script, be sure that the directories that will be created have restricted read access, either by having umask set to 27 if you are creating unix directories, or by having the base directory set for reading only by you if you are creating AFS directories. There is a script called *setacl* that will reset all the AFS directories to the correct permissions in case you get them wrong.

5. Getting an initial set of sources

5.1. machtree

The recommended way to get a copy of the kernel sources is to use the *machtree* script. This method is particulary suited to members of the kernel development group who wish to have their kernel changes kept as a branch in the RCS tree and eventually plan to merge them into the mainline. This script sets up the enviroment needed to make the *bscripts*, the *kmerge* script and *make* work correctly. See */usr/mach/man/man8/machtree.8* for the complete documentation.

To use *machtree*, first, create an empty directory where you would like to work and *cd* there. Then create an *.BCSconfig* file that describes the version of the kernel sources from which you wish to work. A recommended form of the RCS configuration is based on a time. For example, the following is a valid configuration file:

```
<88/07/13,12:00
```

The file *.../kernel/latest/VERSIONS* contains the time at which the sources corresponding to each new version were deemed stable (i.e., unlocked after the merge process). Alternatively, *machtree* will create a *.BCSconfig* file for you. Use the *-help* switch to *machtree* for a complete list of the options.

machtree uses the directory lists in */afs/cs/mach/src/kernel/latest/Directories* to decide which files you want to have checked out. For example

```
machtree MI vax ca sun3
```

will check out only those files needed for those three machines.[1] In addition to building the directory structure, the *machtree* program will create an *RCS* symbolic link in each directory that refers to the appropriate directory in the real RCS tree. At the top level *RCS* will be a link to */afs/cs/mach/rcs/kernel* and at the lower levels, for example, *kern/RCS* will be a symbolic link to *../RCS/kern*. Similarly, symbolic links will be made for *.BCSconfig*, referring to the top-level file *.BCSconfig*. It will also create a copy of the Directories directories at the top level. These files will allow the normal RCS utilities to be used in any subdirectory, but will collect snapshot or configuration information in the top-level directory.

After all the files are checked out, *machtree* creates the necessary links for *machine* and *mach/machine* directories. If your working directory is on a local file system *machine* links to *vax, ca* or *sun3* . If it is on AFS the directory *MACHINE* is created and links are made in it to *../vax_mach, ../ibmrt_mach* and *../sun3_mach*. *machine* is then linked to *MACHINE/@sys*. If the working directory is on AFS the *OBJS/{ibm032,vax,sun}* directories are created to hold the object files. Then an appropriate *Makeconf* file is created.

5.2. cupdate

If you want a copy of a specific version of the kernel sources, but are not intending to put any changes back into the rcs tree, you may perfer to use the *cupdate* script. This script is driven off the snapshot files in *.../rcs/kernel/SNAPSHOTS*.

To generate a complete system checkout of the Xnn version for three machine types to the current directory run the following command:

```
cupdate -sXnn MI ca sun3 vax
```

This will generate an rcsco script file in Xnn.upd. If one *cds* to */afs/cs/mach/rcs/kernel* and runs it, all the files will be checked out. This method does not create all the RCS and .BCSconfig links in your source area. It will, however, setup the machine links and the *OBJS* directories.

cupdate is more useful when you wish to update your copy of the sources from one version to another. For example, if your current set of files corresponds to X40 plus your own changes and you wish to

[1]The *machtree* program currently produces a message from the *find* program about each directory for which you do not have AFS permission. If you don't need access to those (machine-specific) directories, just ignore the messages. To get AFS access, you'll have to send mail to Gripe; they may require that you sign some confidentiality agreements as required by our source licenses.

update to X43 run

```
cupdate -sX43 -sX40 MI ca ...
```

This will create an rcsco script in *X43.upd* that will checkout the files that have changed between X43 and X40. Note that this set of files should correspond to the union of the files in *.BCSset-TRUNK* for X43 and X40 for the machine types that you have requested. You can *cd* to */afs/cs/mach/rcs/kernel* and run this script to check out all the files that are different, or you may want to compare these files with the ones that you have changed. When an update script is being generated, *cupdate* does not create any of the machine or OBJS directories and links.

5.3. sup

If you do not have access to AFS, but want to get a copy of either the **latest** or **alpha** release of the kernel sources, it is still possible to *sup* the kernel sources from *wb1/usr/kernel/{latest***|***alpha}*. To do this you must set up the proper *sup* files on your target machine, and you will need to add a release for the machine to which you wish to sup. To do this you must place the files <release>.host and <release>.list in *wb1/usr/kernel/{latest***|***alpha}/sup/mach.kernel, and place an appropriate line in the releases file in that directory. This method has the least flexibility in terms of the release that you can select and in the information it saves for subsequent updates. Also it will not create any of the extra files and links.*

If you are unable to directly access the /usr/kernel area on wb1 by logging in on that machine as user kernel, you may send mail to machlib@wb1 and request sup access to the kernel sources. Please state what collections you wish (eg. what machine types you are building for and what the name of the machine to which the *sup* is to be made. You will receive a sample supfile containing the crypts for your selected collections, plus some instructions on how to use *sup*.

6. Selecting a configuration

Kernels come in many shapes, sizes, and with different sets of features. Before building a kernel binary, you must decide on a configuration that describes the options you want.

Configuration names consist of groups of symbols separated by "+" or "-" signs. These symbols are then used to select configuration lines from the files in *conf/MASTER{,.local}{,$machine}*. Symbols in capital letters are expanded according to the master files into groups of other symbols, until no further expansions can take place.

A vanilla Mach research configuration is described by the symbol *MACH* for all architectures. From there, you may add or subtract options as you desire. For example, to get a kernel with the experimental emulation code, you would use a configuration called *MACH+me*.

The Facilities staff uses more complicated names in order to better match options and sizes to particular classes of machines. [The *MACH* configuration tends to cause all known device drivers, large kernel tables, and a maximum number of processors to be configured.] A quick primer on the naming scheme used:

- The symbol *STD* is used as a basis for all non-custom kernels. It will build a kernel with small table sizes, which can be modifed if necessary by patching the kernel.
- Multiprocessors are configured to have *16* or *32* processors. Devices specific to

multiprocessors are available by including the symbol *MP*.

- Device support options include: *MF*, for mainframe hosts only; *WS*, for workstations only; *MP*, for multiprocessor hosts; *ANY*, for any host other including those with "special" devices; Only one of these options should be necessary.
- There are also a number of experimental kernel features which appear and disappear frequently. The symbol *EXP* selects the currently useful set that are believed to be working.

For example, a usable configuration for an IBM RT/PC might be *STD+WS*, while a more suitable configuration for a multiprocessor Vax would be *STD+ANY+16*.

The standard configurations of the Mach 2.5 release include the in-kernel nfs and afs code. Since this code cannot be shipped to external sites unless they have the necessary licenses, it may be necessary to build without this code. This is done by specifying -nfs-afs after your configuration name.

7. Compiling a kernel

In order to build a kernel, there must be a directory in which object files should be created. The line begining with *OBJECTDIR=* in the file *Makeconf* in your source directory must set *OBJECTDIR* equal to that directory. *machtree* will create the *OBJS/{vax,ibmrt,sun}* directories for you and edit *Makeconf* to use them with the following assignment:

```
OBJECTDIR=../OBJS/$(machine)
```

where *make* will set *$(machine)* to *vax,ibmrt* or *sun* depending on what machine type you are executing on.

If you did not use *machtree* to get your sources you must do this yourself. If you are only building the kernel for one machine type, you need only create one object directory and set *OBJECTDIR* to it.

Now, merely run *make* specifying the configuration you wish to build:

```
cd my_kernel_sources
make CONFIG=MACH+my_features
```

The command

```
make buildconf
```

will build a standard set of kernels. At the moment it builds three configuations for the sun and rt and five for the vax.

The make command with no arguments will build whatever configuration is defined by a CONFIG= line in Makeconf.

8. Making modifications

When you wish to make changes that can be stored in the RCS tree, you will need to use the bscripts. These scripts require a number of implicit parameters which can either be kept in files in your source area, or in environment variables or input as explicit parameters to each command. See bci(1) for a detailed description of the bscript commands.

The one file that you must have in the base of your source area is: *.BCSconfig-<branch_name>* which contains the configuration rules used to select files that are not on your branch. The following parameters can be either files or environment variables: *.BCSBRANCH* file or *BCSBRANCH* environment variable that specifies the RCS branch on which you would like to store your modifications: a *.BCSVBASE* file or *BCSVBASE* enironment variable which contains the pathname of the RCS source tree. There are also some parameters that must be environment variables. *BCSBBASE* which must be the complete pathname of the base of your source files; *RCS_AUTHCOVER* which is the program to use for authentication (*/usr/misc/.sdm/lib/authcover*), *AUTHCOVER_USER* which is the user that has access to the RCS tree (kernel); *AUTHCOVER_TESTDIR* which is the base of the RCS tree (/afs/cs/mach/rcs/kernel). *machtree* will create the file variables.

Using the program /usr/cs/misc/.sdm/bin/workon with a project description will create all the necessary files and environment variables listed above. *workon* will, however, insist on a Makeconf file existing in the source_base directory.

A sample project description that could be used is:

```
project kernel
project_base /afs/cs/user/<userid>/kernel
rcs_base /afs/cs.cmu.edu/mach/rcs/kernel
rcs_cover /usr/misc/.sdm/lib/authcover
rcs_owner kernel
source_base .
object_base obj/@sys
```

Place the project description on your LPATH with the name project/kernel. Then the command

```
workon -project kernel -branch mrt_X50
```

will exec a subshell, set all the necessary environment varibles, and cd to the source base directory. The first time that *workon* is called it will create the following files:

.BCSVBASE containing the line

```
/afs/cs/mach/rcs/kernel
```

.BCSconfig-mrt_X50 containing the line

```
<88/07/13,12:00
```

where the time will be the time *workon* was first run. You may need to modify this date to correspond with the creation time of the kernel version that you want your branch relative to.

.BCSconfig-TRUNK containing the line

```
<>
```

.BCSpath-mrt_X50 containing the line

```
.
```

.BCSlock containing locking information.

The various *bscripts* will maintain the *.BCSset-<branchname>* file and an *.BCSlog-<branchname>* file that contains your RCS log messages as you check in your changes. Note that the branch name must begin with your user name (*$USER*); you may use just that, or add a suffix.

If you are not using *workon*, you must set the *BCSBBASE* environment variable to the full path name of your kernel source directory, as would be returned by *pwd*. The easiest way to do so is:

```
cd my_kernel_sources
setenv BCSBBASE 'pwd'
```

You must also set all the AUTHCOVER variables by hand.

Then, you may check out sources from anywhere in the source tree, using the *bco* command. For example,

```
cd my_kernel_sources
bco vax/pmap.c
cd bsd
bco kern_exec.c
bco ../vm/vm_map.c
```

The first time you run one of the *bscripts*, you will be asked to authenticate as the *kernel* AFS userid.

9. Checking in changes

In order to checkpoint your work, or to share your work with others, you may wish to periodically check in your changes to the RCS tree. To do so, you run the *bci* program. For example:

```
bci -[switches] kern/mach.defs
```

If the *bci* program is run with no switches it will look for a whist style log entry between the HISTORY and the $Log:$ line in your file to be used as an RCS log message. If there is no such entry, an editor will be invoked, primed with the differences between your source file and the version from which you began.

If the file being checked in does not have a comment line containing the phrase *Log*, the *bci* program will insist that you edit the source to include one. A line should be added after the *HISTORY* comment; that line should begin with the appropriate comment leader, then *Log*.[2]

After an adequate log message is ready, the *bci* program will allow you to check the file in, make further edits, or abort the checkin procedure.

If the file you are checking in did not exist in the source tree before, the *bci* program will ask you whether an original source file from another software vendor (e.g., Berkeley, Sun, DEC) is available, and if so, where the source file may be found. If your file is original to Mach, you should specify that it is "local". If an original is provided, it will be checked in as revision 1.1. Local versions begin with revision 2.1.

10. Updating your branch

Often a developer may keep his changes on a branch for a considerable length of time before merging them back into the mainline. As time goes by, it is a good idea to bring the branch up-to-date with changes that have been taking place on the mainline.

---

[2]Hopefully, most source files will have these lines very soon, so this shouldn't be an issue. In the near future, you may find yourself adding *Log* lines to a number of files.

The easiest current procedure for updating an existing branch, for example, mrt_X92, to the current sources, X99, is:

1. Be sure that all changed files are checked in on the old branch, mrt_X92.
2. Use "cupdate -sX99 -sX92 <dirlist>" to update your sources to X99 or use "machtree <dirlist>" to check out a completely new X99 tree.
3. Do a "workon -project kernel -branch mrt_X99" to create the new branch and set you up to be working on it.
4. Use a "bmerge -rmrt_X92 <filelist>" to merge the old branch into the new branch.
5. Use a "bci -xlog mrt_X92 <filelist>" to check the merged files into the new branch, doing the right thing with the log messages.
6. Suspend the workon for branch mrt_X99. Enter a workon for branch mrt_X92 and remove the old branch with "bcs -o1- <filelist>". This removes all the old revisions for branch mrt_X92, but still leaves the symbolic name.
7. Exit the workon for branch mrt_X92. Delete any extraneous .BCS*X92 files in your source area. Do all further work on the mrt_X99 branch.

11. Merging your changes back to the mainline

The *kmerge* script must be used to merge your changes back to the mainline. Use of this script ensures that all the steps specified in Section 2 take place, leaving the sources in a consistent and predictable state.

For the *kmerge* script to execute correctly you must:

- Be running with userid of *kernsrc* and groups of *systems* and *kernel* To do this

  ```
  nu kernsrc
  Password: < type local password for kernsrc >
  ```

  and be sure that kersrc is a member of the *kernel* group which allows execution of the reverse SUP to wb1.

- Be authenticated as *kernsrc* to AFS.

  ```
  log kernsrc
  Password: < type AFS password for kernel >
  ```

- Have your USER environment variable set to your own userid. For example:

  ```
  setenv USER mywoung
  ```

- Have the rcs commands, /usr/misc/bin, on your PATH.
- Be executing on a machine that is able to *sup* the kernel sources to wb1. This requires an entry for the machine in */wb1/usr/kernel/latest/sup/kernel*. These files are *suped* from /afs/cs/mach/kernel/latest/sup/kernel. To add a machine to this list, copy one of the existing <HOSTNAME>.sup files giving it your HOSTNAME, and then edit the *host*= item to have your machine name. The file *.../latest/sup/xpatch.host* must also be edited to have a line it in for your machine. These files should then be checked in to the rcs tree. These files must be *suped* (or otherwise copied) to wb1 before a *sup* from your host will work.
- The */afs/cs/mach/kernel/latest/.BCSlock* must either not exist, or have been locked by you on the previous *kmerge* pass.

kmerge is called at least twice while doing a check-in. The first time it is called you must give it a list of files to be merged. These files should already be checked-in to the rcs tree with a single branch name.

For example:

```
kmerge [-auto] ‘cat RCSset-mystuff‘
```

kmerge will check the status of the lock by comparing the value of $USER to the name in the file if it exists. If the file does not exist, *kmerge* creates it, locks it for you and gives you a chance to put a comment in the file to say what you are changing. It will then prompt you for the branch name of your changes. It will do a *bgraft* with that branch, copying over and compacting the log messages, and giving you a chance to edit conflicting versions. If *kmerge* was called with the switch *-auto*, the files will be automatically checked in if there are no conficts. If the *-auto* switch is not used, you must interactively confirm the checkin of each file. During this process the *.BCSlog-<branch_name>* and *.BCSset-<branch_name>* files are created. *.BCSlog-<branch_name>* contains log messages for all the files that are being merged. *.BCSset-<branch_name>* contains a list of all the files that have been merged. At the end of the check-in process, all the latest merged versions are left checked out in *.../latest/kernel*.

Once your changes are merged and the resulting files checked out in latest, the first pass of *kmerge* exits and you are expected to build the kernel for a vax, sun and rt using the latest sources. It is assumed that you will build in /afs/cs/mach/build/{ibmrt_mach,sun3_mach,vax_mach}/latest/kernel. You should decide if you want to build incrementally on top of the last kernel build or if you want to delete everything in that area and build from scratch. You need to be authenticated as *kernbin* to write in this area. The kernel Makefile uses an environment variable, *RELEASE*, and the *make* variable *machine* to chose the appropriate build area. If this *RELEASE* is not set or set to *latest*, /afs/cs/mach/obj/@sys/kernel/latest will be used for the build area.

To build the standard kernel configurations;

```
log kernbin
Password: <type kernel AFS password >
<Decide if you wish to delete existing stuff from
 /afs/cs/mach/obj/{ibmrt_mach,sun3_mach,vax_mach}/kernel/latest.>
cd /afs/cs/mach/kernel/latest
make buildconf
```

This command builds one large, many featured, kernel for each machine type. The binaries are put in */afs/cs/mach/build/<machine>latest/kernel/STD+ANY+EXP[+16]* If you wish to try out other configurations, for example the MACH configuration with the experimental features turned on, just type

```
make CONFIG=MACH+EXP
```

The kernels that have been built should be booted onto an appropriate machine and tested. A generic test script can be found in */afs/cs/project/mach/root/tests/mach_tests*. One should also try out the user programs that depend on kernel structures. They include: *ps, w, top, pstat, netstat, iostat adb, dbx* and *gdb*. If you have changed the *proc.h, user.h, inode.h* or network structures, in a way that breaks any of these programs, you should either reconsider your change, or increment the value in *sys/version* and build all the kernel-dependent user level programs with your new include files.

If bugs are found during the testing process, *kmerge* should be called again with a new list of files to be merged. This process is repeated until the kernel has passed its tests.

Once the kernel has been tested, you execute *kmerge* again, this time with no arguments. It now checks for any files that you may have changed in the ../kernel/latest area and asks you if you want to check them in. It then creates the snapshot files, does the *sup* to wb1, and prompts you for a post to the *mach-kernellog* bboard. At the end it checks in the *.BCSset-TRUNK* file and removes the *.BCSlog-TRUNK* and *.BCSlock* files. Since several of these steps are time consuming and occasionally cause the script to abort, the script will ask you if you have already successfully completed each step. None of the steps are optional and if any files have been changed, the snapshots and the sup must be redone. However, if you are rerunning the script for some other reason, you do not have to do these steps twice.

The only things that are not currently done for you by *kmerge* is to update the Directories list, in case you have created or deleted any directories in *kernel* and to check for an include version update.

Table of Contents