CMU-tools for Mach 3.0 Development

Mary R. Thompson

29 June 1992

1. Introduction

This document describes the organization of the Mach 3.0 sources and build areas and the methods for changing them. It documents the standard method for getting a private copy of selected sources: the *m3tree* script; the building your own version of the Mach 3.0 programs, and the required way of inserting your changes back into the official sources: the *m3merge* script. It is intended to be used by the Mach 3.0 developers at CMU.

2. Background knowledge

2.1. Andrew File System

The Andrew File System (AFS) is used to hold all of the sources and compiled object files. An understanding of the AFS filesystem in general, and its protection scheme in particular, is required.

CMU systems recognize the special name *@sys* in symbolic link names. When evaluating such a link name *@sys* is bound to *vax_mach, rt_mach, sun3_mach, sun4_mach, i386_mach, pmax_mach* or *luna88k_mach* depending on what type of machine the task is executing on.

Most AFS volumes are backed-up in two ways: the /afs/cs/.BACKUP directories and a nightly backup to tape. However, some volumes are set up without such backup to hold object or other easily regenerated files. These volumes can be identified by doing *"fs lv ."* on a directory and getting the message "*** No backups made for this volume ***". The offical Mach object areas are on such volumes and there are several other such volumes for individual use.

2.2. Make

The local version of *make* has been modified to deal with the problem of building programs for several 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 *target_machine* is set to *vax, pmax, i386, luna88k, sun3* or *sun4* depending on what machine type the *make* is being executed on. The *TARGET_MACHINE* variable has the same values only in upper-case. *OBJECTDIR* is typically defined in a file called *Makeconf* as a function of *target_machine* and relative to the source tree. When *make* is run with the ***-m*** switch it will search for files in subdirectories named *TARGET_MACHINE* before looking in the curent directory. *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. Another switch to remember is ***-c***. This will prevent *make* from automatically looking for RCS files for a prerequite file, if it can't find the prequiste file. This is mostly important when *make* is set up to search lots of directories. The ***-d*** switch will procduce lots of information about what it is doing. See *make(1*) for more complete information.

2.3. 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 mostly used to provide copies of our files to external sites. You can sup files locally, but the machine that you are supping to, must be in our host tables for the Mach 3.0 collections on x29. For more information see *sup.1* or */afs/cs/project/mach/public/sup/sup.doc*.

2.4. RCS/BCS

RCS is used extensively to maintain a history of modifications to the Mach sources, and to aid in managing concurrent lines of development. Therefore, a working knowledge of RCS is necessary. 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 have been introduced to facilitate the specification of which versions to chose for 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 *b-commands* 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. The *b-commands* need various bits of information before they can function. Most of these parameters can be supplied by environment variables, command line switches or files. Our standard tools set one environment variable, *BCSBBASE*, which defines the base of your shadow or working area and then use files within that area to define the other values.

The following *.BCS-files* are of interest:

.BCSBRANCH specifies the RCS branch on which you would like to store your modifications

.BCSVBASE containing the pathname of the RCS source tree.

.BCSconfig-<branch_name>
: which contains the configuration rules used to select files that are not on your branch.

.BCSset-<branch_name>
: Whenever you either *bco* or *bci* a file, its name is added to this file. Thus this file gives a complete list of all files that you have modified.

.BCSlog-<branch_name>
: Whenever you *bci* a file, its name and log message is placed in this file.

.BCSlock This file locks a source or shadow area. It's purpose is to ensure that only one user at a time is using the *b-commands* in the same working area.

Further information can be found in the the man page for *bci(1)*.

2.5. workon

/usr/misc/.sdm/bin/workon is a command that reads a project description file which defines the source, rcs and possibly shadow area, creates any missing *.BCS-files*, and leaves you in a subshell with *BCSBBASE* set as well as a few other BCS variables.

A sample project description that could be used to define a working area for the mk sources with no shadowing is:

```
project mk
project_base /afs/cs/user/<userid>/mach3
rcs_base /afs/cs/project/mach3/rcs/mk
source_base src/mk
object_base obj/@sys/mk
```

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

```
workon -project mk -branch mrt_MK73
```

will exec a subshell, set all the necessary environment variables, and cd to the source base directory. Assuming that <*userid*> was set to *mrt*, *workon* will create in */afs/cs/user/mrt/mach3/src/mk* any of the following files that do not already exist:

.BCSVBASE containing the line

```
/afs/cs/project/mach3/rcs/mk
```

.BCSconfig-mrt_MK73 containing the line

```
<current date and time>
```

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_MK73 containing the line

```
.
```

.BCSlock containing locking information.

If you wanted to shadow the user sources you would use a project description like:

```
project user
shadow_base /afs/cs/usr/mrt/mach3
shadow_source_base src/user
shadow_object_base obj/@sys/user

project_base /afs/cs/project/mach3
rcs_base rcs/user
release_base latest/release/@sys

# change the next line to shadow from another area
source_base latest/src/user

# this object_base is just to appease workon which insists
# that an object_directory exists, even if it is never used
object_base /afs/cs/project/mach-mrt/mach3/obj/@sys
```

2.6. Other

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

The Mach 3.0 specific scripts that are mentioned in this document can be found in */afs/cs/project/mach3/tools*. The best versions of the *RCS* programs and *b-commands* 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. There are also symbolic links to these programs in */usr/misc/bin*. *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 Mach 3.0 source, release and build trees

All the Mach 3.0 tools, source, object, and release area can be found starting at */afs/cs.cmu.edu/project/mach3*. There are currently 3 versions there: *merge* where merges take place, *latest* the latest official version of sources and releases, and *alpha* which is supposed to be an older more stable version of the sources and releases. The tools directory is where we keep useful scripts.

The source, object and release directory organization relative to /afs/cs/project/mach3/{merge,latest,alpha} is:

```
src/{mk,ux,user,poe,us,bsdss,mdos}/...
obj/{pmax_mach,i386_mach,...}/{mk,ux,user,...}/...
release/{pmax_mach,i386_mach,...}/{bin,lib,include,etc,man,special}/...
```

mk is the kernel directory; *ux* is the Unix-server area (BSD 4.3 flavor); *poe* is the Poe server; *user* is the area for user level programs: *us* is the area for the multi-server work; and *mdos* is the Mach-dos server; and *bsdss* is the BNR2 based license-free Unix server.

Each time the latest area is updated some information about the release is created:

- The file *{mk,ux,user,poe,us,bsdss,mdos}/conf/version.edit* is incremented by one and checked into the RCS tree.
- A post is made to the mach3 mailing list with a summary of all the changes to this version. This post is also saved in the MERGE_HISTORY file in the sources.

4. Kernel AFS groups and privileges

If you are new to AFS, be aware that unprotected files on AFS can be read by everyone at CMU and at quite a few other sites. Check the contents of /afs to get an idea of how widespread access is. To help people limit access to useful subsets of the world there are several standard access groups provided by our AFS cell administation. Some of them are:

system:anyuser	anybody at all
system:authuser	anyone who is authenticated in the CS.CMU cell
system:friendlyhost	anyone, authenticated or not, referencing the file from a CS-CMU machine.
system:campushost	anyone, authenticated or not, referencing the file from a CMU machine.

Mach project files that contain no licensed code are generally readable by system:anyuser or system:authuser and system:friendlyhost, depending on how proud we are of the files. The mach rcs tree is restricted for reading to the kernel:view.local group. This is to protect any old licensed code that may be in previous versions of the files.

Read access to the Mach 3.0 sources that contain licensed code must be much more carefully controlled. We use the facilities controlled groups kernel:view.{4bsd,sun3,pmax,nfs} and sys:read.afs to grant read access. The kernel:view and system:view groups are in the process of being renamed to sys:read groups. To be added to any of these groups, send mail to help@cs.cmu.edu specifiying which groups you want to be added to.

Files in directories that are protected with these read groups must never be copied to AFS directories with greater permissions or to local directories with world-read access. (Remember that at CS-CMU, RFS allows users from other machines to read files on your local machine). Most of the licensed code lives in the *ux* area, but there is also some in the *mk/kernel/{vax*,sun*}* directories.

If you are going to check out files from the kernel sources using *m3tree* 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 local directories, or by having the base directory set for reading only by you if you are creating AFS directories.

Write access to the sources and RCS files are controlled by a different group for each collection: mach:update.{kernel,unixserver,poe,user,mdos,us}. If you want to check in files on a private branch in the RCS tree or do a merge, you will need to be in the appropriate group. To be added to these groups send mail to mach@cs.cmu.edu speciying what access you need.

5. Getting an initial set of sources

5.1. m3tree

If you are at CMU and have read access to the mach3 archive area, you can get a copy of the sources by using the ***/afs/cs/project/mach3/tools/bin/m3tree*** script. You should run this script either in a directory *<base>/src* or *<base>/src/<collection>* where *<collection>* is the subtree you are checking out. The arguments for this command are:

```
m3tree [-options] [mk|ux|user|poe|us|bsdss|mdos] [<Dirlist>]
```

where the options are:

`-dir <dir>`	specify a dir to process. Can be used if you only want to get the files from a few directories.
`-co`	check out subdirectories (default)
`-noco`	don’t check out subdirectories. Can be used to set up an empty tree with the RCS links and .BCS files in place
`-q`	be quiet
`-v`	be verbose (default)
`-i`	ask questions (default)
`-f`	don’t ask questions; use default answer
`-b <version>`	specify a base version. Can be used if you want to check out an older version than the one currently in latest. <version> is of the form MK70 or UX28.
`-n <name>`	specify a branch name. Can be used to get the files off a branch rather than the

trunk.

`-help` this summary

The ***Dirlist*** names correspond to the files in ***{mk,ux,...}/Directories*** directory. e.g. *ALL, MI, ca, sun3, i386* ... *ALL* will select all the files for all platforms. *MI* gets the machine independent files to which you can add the specific platforms for which you want to build. Thus

```
m3tree mk MI mips
```

checks out all the sources needed to build the latest kernel for the DecStation.

```
m3tree -b UX30 ux MI sun3 vax
```

checks out the files needed to build the UX30 Unix server for a sun3 and vax.

Note: if you want to checkout sources earlier than MK68 or UX29 you need to use the *mktree.obsolete* and *uxtree.obsolete* scripts as the directory layout changed at that time.

In addition to building the directory structure, the *m3tree* program will create all the *.BCS-files* that the *b-commands* need and 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/project/mach3/rcs/{mk,ux,user..}* and at the lower levels, for example, *kern/RCS* will be a symbolic link to *../RCS/kern*. 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 configuration information in the top-level directory. Note that if you do use the plain RCS commands, you will not get entries in the *.BCSset-<branch>* or *.BCSlog-<branch>* files that the *b-commands* create that allow for automatic tracking of all the files that you change.

5.2. sup

If you do not have access to AFS, but want to get a copy the **latest** release of the kernel sources, it is possible to *sup* the kernel sources from *x29:/usr2*. To do this you must set up the proper *sup* files on your target machine, and you need to have the machine onto which you wish to copy the files added to the host tables on x29. If you want to sup into an AFS directory, you can use the machine ernst.mach.cs.cmu.edu as sups are already enabled to it with the crypt *testing*. For futher information see the file */afs/cs/project/mach/public/doc/mach3/mach3_sup*.

5.3. Shadowing sources

It may be that you do not want a complete copy of a source tree. For example if you are only changing a few files in the user area, or want to do a quick build of some version of the kernel or other server changing none or only a few files. (The emphasis here is on quick as there is no guarantee that the sources in the latest area will stay the same for any period of time.) Our *make* and *Makefiles* are set up so that it is possible for *make* to look in both a ***shadow*** and a base source tree for files. Any file found in the *shadow* will be used. If there is no file in the *shadow* the one in the base tree will be used.

Setting up a shadow directory so that you can use the *b-commands* can be done in either of two easy ways: you can use *workon* with an appropriate project description file, or you can use *m3tree* with the *-noco* option. If you chose to use *m3tree*, you must remember to set *BCSBBASE* and *BCSBRANCH*.

Do either:

```
workon -project <project_name> -branch <branch_name>
```

or

```
cd  <your shadow src directory>
m3tree -noco <project_name>
setenv BCSBBASE ‘pwd‘
```

to create all the necessary *.BCS-files* in your shadow user area and to set the environment variable *BCSBBASE*.

The *m3tree -noco <project_name>* only needs to be done once, but *BCSBBASE* must be set whenever a *b-command* is executed. If you want to work on more than one branch in the same working area, you do a *workon* for whichever branch you want to currently workon. If you want to do this without using using *workon*, you can set the environment variable *BCSBRANCH* after using *m3tree* with the *-n* option to set up the *.BCS-files*.

If you are planning to do build in your shadow you should check out the Makeconf file. That is the one file that *make* will not find in the shadow area. You also need to tell *make* where the master source directory is. One way is to add the line

```
SOURCEDIR=/afs/cs/project/mach3/latest/src/<collection>
```

to your local copy of the *Makeconf* file. This make variable can also be set on the command line to make.

6. Making modifications

When you wish to make changes that can be stored in the RCS tree, you will need to use the *b-commands* to check files in and out of the RCS tree. See the previous section on RCS/BCS.

You may check out the files you want to change 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
```

In order to checkpoint your work, to share your work with others, or just to update the *.BCSset* file so that you will have a list of all the files you have modified, you may wish to periodically check in your changes to the RCS tree. Also it is easiest to merge your files onto the mainline if they are all checked into a branch before you start the merge process.

To checkin a file, 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*.

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.

There are several other ways that *bci* can find a log entry: there is the *-xlog* switch and the *-m"put log here"* switch. See the *bci(1*) for more details. Generally, you should always create a *whist* entry when you are editing the file in the first place, then it is not a major nuisance to create them at check in time and more importantly there will be a record of what changes have been made to the file.

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.

7. 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.

The easiest current procedure for updating an existing branch, for example, mrt_MK70, to the current sources, MK73, is:

1. Be sure that all changed files are checked in on the old branch, mrt_MK70.
2. Use "m3tree mk <dirlist>" to check out a completely new MK73 tree.
3. Do a "workon -project kernel -branch mrt_MK73" to create the new branch and set you up to be working on it.
4. Use a "bmerge -rmrt_MK70 <filelist>" to merge the old branch into the new branch.
5. Use a "bci -xlog mrt_MK70 <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_MK73. Enter a workon for branch mrt_MK70 and remove the old branch with "bcs -o1- <filelist>". This removes all the old revisions for branch mrt_MK70, but still leaves the symbolic name.
7. Exit the workon for branch mrt_MK70. Delete any extraneous .BCS*MK70 files in your source area. Do all further work on the mrt_MK73 branch.

8. Merging your changes back to the mainline

8.1. M3merge: pass 1

The */afs/cs/project/mach3/tools/bin/m3merge* script must be used to merge your changes back to the mainline. Use of this script ensures that only one person at a time is doing a merge and leaves the merge and latest areas in a consistent and predictable state.

For the *m3merge* script to execute correctly you must:

- Be authenticated to AFS and a member of the access group mach:update.mach3 and the appropriate mach:update group for the area you are merging.
- Have the rcs commands, /usr/misc/bin, on your PATH.
- The */afs/cs/project/mach3/merge/src/{mk,ux,..}/.BCSlock* must either not exist, or have been locked by you on the previous *m3merge* pass.

m3merge is called at least twice while doing a check-in. The first time it is called you normally 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:

```
m3merge mk|ux|user|poe|mdos|bsdss ‘cat .BCSset-mystuff‘
```

m3merge 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, *m3merge* 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 *bmerge* with that branch, copying over and compacting the log messages, and giving you a chance to edit conflicting versions. During this process the *.BCSlog-TRUNK* and *.BCSset-TRUNK* files are created. *.BCSlog-TRUNK* contains log messages for all the files that are being merged. *.BCSset-TRUNK* 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 */afs/cs/project/mach3/merge/src/{mk,ux,user,poe.bsdss}*. Note that *TRUNK* is a branch name that the *b-commands* recognize to mean to check things on or off the RCS trunk.

Once your changes are merged and the resulting files checked out in merge, the first pass of *m3merge* exits and you are expected to build the tree for an i386, pmax, vax, and sun3 and using the sources in merge.

8.2. Making changes in the merge source area

If you want to make any changes directly to files in the src/merge area, you should use *bco*, to lock the rcs file on the trunk, and *bci* to check in your changed file. You must set the environemnt variable, *BCSSBASE*, before the *b-commands* will work. Do this either by

```
setenv BCSBBASE /afs/cs/project/mach3/merge/src/<collection>
```

or

```
workon -project <collection>.merge -branch TRUNK
```

The merge project descriptions can be found in */afs/cs/project/mach3/tools/lib/project*.

8.3. Adding to Directories

At this point if you have added any new directories to the tree you should add them to the appropriate file in the *Directories* directory. The new directories should be added to the *ALL* file and either to the *MI* (machine independent) file or the machine(s) file in which it belongs. *bco* the file and update it. You can either check it in manually or wait until the second part of the *m3merge* process to do it for you.

8.4. Defuncting files and directories

If you have deleted any files or moved files to a different directory you need to defunct the file in the rcs tree. This keeps the file from being checked out by either the *m3tree* script or by the second pass of *m3merge* when it checks files out to the latest/src area. The second pass of *m3merge* will in fact delete files from latest that have been marked defunct in the rcs tree. To defunct a file you need to lock the file and then checkin a null file with the state set to defunct. A procedure for doing this is:

```
bco <file>             -- locks the file
cp /dev/null <file>    -- creates a zero length file
bci -defunct -m"Deleted" or -m"moved to ..." <file>
                       -- checks in the file and sets
                       -- the state to defunct
rm -f <file>           -- gets rid of the null file
```

If you have removed all the files in a directory, you need to also manually remove the directory from the src/merge area and to remove it from the appropriate *Directories* files. After the copy from merge to latest is made, the directory also has to be removed from src/latest. Note that it should be empty by this time, if all the files were successfully defuncted. The directory and files must remain in the rcs tree so that older versions of the kernel sources can still be retrieved.

8.5. Building and testing

Now you are ready to build. It is assumed that you will build in /afs/cs/project/mach3/merge/obj/{i386_mach,pmax_mach,vax_mach,sun3_mach}. You should decide if you want to build incrementally on top of the last build or if you want to delete everything in that area and build from scratch. The project machines: wb1 (sun3), natasha (vax), ernst (pmax) and intel2 (i386) may be used to build on.

See the document **Building Mach 3.0** if you have any questions on how to build a collection. Basically at this point the way to build and release poe and the user tree is to simply execute the make command at the top of the tree you are building, e.g. /afs/cs/project/mach3/merge/src/user. This will build in ../../obj/@sys and release the files to ../../release/@sys.

For the kernel and Unix server it is recommended to use the m3build script to get all the configurations built. This script should be run in /afs/cs/project/mach3/merge/src as follows:

```
m3build mk|ux -extra
```

The *-extra* switch to *m3build* is used to get all the officially supported versions of the kernel and Unix server built.

If bugs are found during the testing process, you can use bco (to lock the version on the trunk), correct the file and bci it, all in the context of the merge directory. This process is repeated until the system has passed its tests.

8.6. M3merge: pass 2

Once the system has been tested, you execute *m3merge* again, this time with no arguments. It now checks in the new files to the TRUNK. Then it checks for any files that you may have changed in the ../merge/src area and asks you if you want to check them in. After any remaining files are checked in, *m3merge* prompts you for a release message to send to the *mach3* mailing list. It then appends the release message to MERGE_HISTORY file and checks that file in.

If this is a merge being done in the merge area, *m3merge* checks out all the files in *.BCSset-TRUNK* into the latest area. After it does this it asks you if you want to run diff on the merge and latest areas. They should be identical at this point except for the .BCS files. Sometimes if there have been errors and not all the changed files in the merge area have been checked in, these areas will differ. Thus it is a good idea to let it do this diff. The results of the diff are put in */tmp/m3merge.diffs* as well as being displayed on stdout. 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. However, if you are rerunning the script for some reason, you do not have to do these the steps twice.

The only things that are not currently done for you by *m3merge* is to update the Directories list, in case you have created or deleted any new directories, and to move the released binaries to mach3/latest/release. Running

```
make TRELEASEDIR=/afs/cs/project/mach3/latest/release/@sys
```

make will copy the current build targets from ...merge/release to the parallel directories in ...latest/release. Note that this release step must be done on each platform that you have built for. In the case of the kernel, and ux, poe and and bsdss servers the new binaries will have different names from the previous ones and will thus not replace anything. Periodically some of the old versions of these servers need to be removed to keep the release areas a resonable size. Releasing the user files and the kernel include and libraries will overide the existing versions. So some thought should be given about moving the latest release programs to alpha before overriding them.

8.7. Deleting your old branch

It is also a good idea once the merge has been done to delete your branch. This keeps down the space used by the rcs tree and keeps the rcs headers from growing beyond their allowed limits. To delete a branch go back to your working area, do a workon for your old branch and run

```
bcs -rm -o1- ‘cat .BCSset-<branch_name>‘
```

The *-rm* switch causes the file to be deleted from the .BCSset file and from your working directory. At this point you would need to do a new *m3tree* command to start doing additional work relative to the current head of the tree.

Table of Contents