Online version: http://dust.ess.uci.edu/doc/vcs March 13, 2017

VCS Cheat Sheet

by Charlie Zender
University of California, Irvine

Department of Earth System Science
University of California
Irvine, CA 92697-3100

zender@uci.edu
Voice: (949) 891-2429
Fax: (949) 824-3256

Contents

1 Software Configuration Management

The process of orderly tracking, versioning, and serving, and administration of project source code is called *Software Configuration Management* (*SCM*). A modern SCM environment requires a

Version Control System (*VCS*). ¡This document describes three different VCS software packages: Concurrent Versioning System (CVS) and Subversion (SVN). These tools are intimately related since SVN is designed by some of the original authors of CVS to replace CVS. Currently, all of Section 2 (99% of this document) is devoted to CVS, while the newer portions, Section 3, is devoted to SVN. There is a tiny bit at the end on Git.

2 CVS

Section 2.4 presents the generic CVS commands. The commands proceed in order of increasing complexity. After that, there are working examples from a number of specific modules.

2.1 CVS Preliminaries

2.1.1 Abbreviations

The following abbreviations are used: CWD means the working copy of the CVS module in and beneath the Current Working Directory; `mdl` = module name; `fl` = file name; `brnch` = branch tag; `vrs` = version name (which can be a branch tag).

2.1.2 CVS Keywords

Only "`cvs commit`", "`cvs tag`", and "`cvs rtag`" change the repository. All other CVS commands change, if anything, the contents of the CWD only, and so are recoverable. CVS commands should be executed from the top level of the CWD. In this case, the module name `mdl` may be optional at the end of the command (because CVS finds the module name by looking in the "CVS" subdirectory `mdl/CVS`). Executing CVS commands from one level above the module requires specification of the module name so that CVS knows where to find `mdl/CVS`.

In addition to the usual RCS keywords (`Header`, `Date`, ...), CVS defines the keyword `Name`. The string "`$Name$`" expands to include the tag specified when the module was checked out, e.g., "`cvs co -r ccm3_5_22 ccm`" produces "`$Name: ccm3_5_22 $`" but "`cvs co ccm`" produces "`$Name: $`" for two reasons. First, no tag was specified with the checkout so the default `ccm` module, i.e., the main trunk, is retrieved. The main trunk has no default tag name, so the keyword value is an empty string. Second, checking out the main trunk never expands "`$Name$`", but checking out a branch, any branch, does. Branch tags are sticky, so a branch always has an associated tag name. The rest of the RCS keywords expand to their usual meanings regardless of whether the module is a tagged version or branch. To explicitly turn off CVS keyword expansion when checking out a module (recommended to avoid unnecessary conflicts due to CVS metadata changes), use the "`-kk`" option instead of the default, "`-kkv`". The `-k` options are *sticky*, meaning they apply to any derived files as well.

2.1.3 CVS Errors

Occasionally CVS does not perform as expected. Two errors are particularly frequent, and easy to fix. The first is that, when asked to commit files to the repository, CVS complains that some subset of the files is out-of-date, and asks you to fix this before it proceeds. This is, in fact, not an error,

but very helpful behavior by CVS telling you that somehow your working directory is out of date with respect to the repository. Some versions of CVS will commit those files that are modified but not commit those that are out of date. Other versions of CVS will not commit any file until all files in the CWD are current. In this case, simply perform a "`cvs update`" before attempting to "`cvs commit`" again. This should solve the problem.

The second error occurs when CVS attempts to modify the repository, e.g., during a "`cvs commit`" operation, and it complains that it has never heard of any of the files, e.g., "`cvsbin commit: nothing known about 'to'`". If this occurs on the CGD Sun network, it might be because you have used simply `cvs` rather than `cvsbin` as the CVS command. Check this and try your command again.

2.1.4 CVS Debugging

When attempting to solve problems with CVS, try using the "`-t`" switch so CVS will trace its execution.

2.1.5 CVS Environment Variables

CVS makes use of several environment variables.

```
export CVS_RSH='ssh'     # Needed for ssh access to NCAR CGD CVS
export CVSROOT=':ext:dust.ess.uci.edu:/data/home/${USER}/cvs'
export CVSUMASK='002'    # Default file permissions for CVS
```

2.1.6 CVS Administration

In order to execute the `cvs admin` command, the user must be a member of the cvsadmin group, if it exists.

```
cvs admin -oREV::
cvs admin -mREV:MSG Replace the log message of revision REV with MSG.
```

It took five years since I began using CVS in 1998 before I needed to learn how to use CVS administrative commands to repair repositories. On July 30, 2003 the laptop I was using ran out of power and somehow wrote binary garbage into every file that `emacs` was visiting. I only lost a few hours of work, and it seemed like time to backup everything I was working on. That made things a little worse because I incidentally committed the binary garbage to CVS. How did I remove the binary garbage from the repository?

```
# Replace corrupt file with latest working backup
cvs update -r 1.65 -p pnp.tex > pnp.tex
# Delete garbage file from repository
cvs admin -o 1.65:1.67 pnp.tex # Method 1: Collapse everything between
cvs admin -o 1.66 pnp.tex # Method 2: Delete a specific version
# Delete garbage file from repository
cvs update -r 1.53 -p prp_mri.tex > prp_mri.tex
cvs admin -o 1.54 prp_mri.tex
```

2.1.7 Repository Relocation

Occassionally a CVS repository moves and one must adjust the repository location in all modules that were checked out from that repository. One method of doing this is simply deleting the old modules and then checking them out from the new location. This works fine unless the module is not synchronized with the repository, i.e., if there is new work that has not been checked in. In this case it may be wiser to edit the repository location in the checked out modules, rather than re-checking out the modules.

```
# Modules in repository on esmf.ess.uci.edu
mdl_lst='aca aeroce aeronet afgl anl anv arese avhrr c caco3 ccm ck cld clm
dmr dot dst elisp erbe ess ess_acc ess_atm ess_bnd ess_ccc ess_lsp
ess_phz ess_prc ess_rdn facts frc fsf grd hdf hire idea idl idx_rfr
igbp igpp improve ipcc job jrn linux lsm ltr map match mk
mny ncep ncl perl phd poetry pr prp rt rvw sdn slr_spc specfun tex
time toms uci www'
mdl_lst='bxm c++ esmf f mie mie_ww ppr_BiZ03 ppr_BiZ04 ppr_FlZ05
ppr_GrZ04 ppr_ZMT04 ppr_ZeK05 ppr_ZeT06b ppr_ZeT06 prp_gtcp
prp_itr sltsbl'
mdl_lst='nco'
for mdl in ${mdl_lst} ; do
  cd ${HOME}/${mdl}
  printf "Updating module ${mdl}...\n"
  for fl in `find . -name Root -print` ; do
    printf "Updating Root name in ${fl}...\n"
#   echo 'zender@goldhill.cgd.ucar.edu:/home/zender/cvs' > ${fl}
#   echo 'zender@nco.cvs.sf.net:/cvsroot/nco' > ${fl}
#   echo 'zender@esmf.ess.uci.edu:/u/zender/cvs' > ${fl}
    echo ':ext:zender@nco.cvs.sf.net:/cvsroot/nco' > ${fl}
  done # end loop over fl
done # end loop over mdl
for mdl in ${mdl_lst} ; do
  cd ${HOME}/${mdl}
  printf "Updating module ${mdl}...\n"
  for fl in `find . -name Repository -print` ; do
    printf "Updating Repository name in ${fl}...\n"
    echo ${mdl} > ${fl}
  done # end loop over fl
done # end loop over mdl

# Modules in repository on goldhill.cgd.ucar.edu
aer -> /home/zender/match_dst/dst
cam2_0_2_dev41_brnchT_dust2
cam_dev
ccm_dst
ccsm2
```

```
ccsm2_2_beta08
clm2_deva_52
clm_dev
crm
match_dst

# Modules in repository on dust.ess.uci.edu
bxm
c++
esmf
f
mie
mie_ww
ppr_BiZ03
ppr_BiZ04
ppr_FlZ05
ppr_GrZ04
ppr_ZMT04
ppr_ZeK05
ppr_ZeT06b
ppr_ZeT06
prp_gtcp
prp_itr
sltsbl

# Modules in repository on sf.net
nco
```

2.2 Modules

To create a CVS repository, use

```
cvs -d ${HOME}/cvs init
```

The following subsections present specific examples for both generic and CGD-related modules. Many of the same examples are used for different modules, so that the examples might differ only in the name of the target module, and sometimes not at all (e.g., "`cvs co`"). The goal has been to keep the examples self-contained for each module, at the expense of document length and redundancy. The examples are drawn from modules in active use in CGD. Table 1 summarizes the three repositories used in the examples.

Software	Machine	Repository Location	Module(s)
C++, mie[1]	dust.ess.uci.edu	/home/zender/cvs	c++ mie
NCO	cvs.nco.sourceforge.net	/cvsroot/nco	nco
CCM, CCM-Dust, CRM	goldhill.cgd.ucar.edu	/fs/cgd/csm/models/CVS.REPOS	ccm ccm_dst crm
MATCH, MATCH-Dust	goldhill.cgd.ucar.edu	/fs/cgd/csm/people/eaton/CVS	match match_dst dst
CAM	goldhill.cgd.ucar.edu	/fs/cgd/csm/models/CVS.REPOS	cam2_0_2_dev41_brnchT_dust2

Table 1: CVS Repositories Used in Examples

Most of the examples do not explicitly specify the repository (with the `-d` argument). This is because most commands are executed within, rather than above, the CWD. Table 1 should be consulted when the repository to use is in doubt.

2.3 CVS Server Issues

A precise discussion of setting up a CVS server is given at http://www.korayguclu.de/index.php?&file=linux.cvs.pserver.xml. The CVS documentation describes the necessary modifications to the internet daemon configuration file /etc/inetd.conf. RedHat Linux uses a more powerful and complex (some call this sophisticated) daemon, `xinetd` (pronounced "zy-net-d"), configured in /etc/xinetd.conf. On RedHat systems, CVS password server services are controlled by a file called cvspserver located in /etc/xinetd.Dachau's.

```
# Others
sudo scp ~/linux/etc/inetd.conf.dust /etc/inetd.conf
# RedHat
sudo scp ~/linux/etc/xinetd.d/cvspserver /etc/xinetd.d
sudo /etc/rc.d/init.d/xinetd restart
```

The CVS documentation specifies how to configure the `port number` (2401), `socket_type`, and `server_args` flags. Internet daemon services (i.e., `inetd` or `xinetd`) should be started automatically, in, say, system runlevel 3.

2.3.1 Read-only Access

Place the readers, writers, and passwd file in the directory $CVSROOT/CVSROOT.

```
cd ${HOME}/cvs/CVSROOT
chmod 644 readers writers
chmod 444 readers writers
% cat > readers
sci_prg
jtalaman
mflanner
% cat > writers
zender
mflanner
% cat > passwd
sci_prg::cvspub
esmf::cvspub
```

Add user cvspub to the system. Do not give the user a home directory or interactive access.

Once read-only services work, outside users may check out modules by logging in as the anonymous user.

```
cvs -d :pserver:sci_prg@dust.ess.uci.edu:/home/zender/cvs login
cvs -d :pserver:sci_prg@dust.ess.uci.edu:/home/zender/cvs co -kk c++
```

2.4 Generic Modules

2.4.1 Import an existing directory

Import files in directory `mdl` to create new CVS module `mdl`. The keywords "zender" (vendor tag) and "mdl-0_1" (release tag) are used for initial module tags.

```
cd mdl
cvs import -m "Imported sources" mdl zender mdl-0_1
cvs -d :ext:dust.ess.uci.edu:/home/zender/cvs import -m "Imported sources" m
cd ..
mv mdl mdl.bck
cvs co -kk mdl
ls -R mdl
/bin/rm -r mdl.bck
```

2.4.2 Import RCS files

Edit the CVS repository to create the appropriate source directories. Make sure all RCS files are unlocked, then copy them into the CVS repository.

```
cd mdl/RCS
rcs -u *
mkdir ${CVSROOT}/mdl
cp .*,v *,v ${CVSROOT}/mdl
cd ../..
mv mdl mdl.bck
cvs co mdl
ls -R mdl
/bin/rm -r mdl.bck
```

2.4.3 Checkout

Checkout module `mdl`. A `-d` argument before the verb specifies the CVS repository to use (instead of `$CVSROOT`). By default, a module `mdl` is placed in the `mdl` directory of the CWD. A `-d` argument after the verb specifies an arbitrary directory for the module. The `-kk` option suppresses RCS keyword expansion (e.g., of "`$Header:  $`"), thereby minimizing the number of conflicts during a future `cvs update`. Note that `co` is a synonym for `checkout`—the two are interchangeable.

```
cvs co mdl     # Expands keywords
cvs co -kk mdl # Does not expand keywords. Use prior to cvs update.
cvs checkout -d drc mdl
cvs -d /fs/cgd/csm/models/CVS.REPOS co mdl
cvs -d /fs/cgd/csm/models/CVS.REPOS co -d drc mdl
```

2.4.4 Add

Schedule file `fl` for addition to the repository. Actual addition takes effect with the next "`cvs commit`" command.

```
cvs add fl
```

2.4.5 Remove

Schedule file `fl` for removal from the repository. Actual removal takes effect with the next "`cvs commit`" command.

```
cvs remove fl
```

2.4.6 Query

Show changes of CWD relative to repository. Option `-n` specifies that no changes to the repository will occur.

```
cvs -n update
```

2.4.7 Diff

Show changes relative to particular versions, tags, or times.

```
cvs diff -kk main.c
cvs diff -r ccm3_6 -kk main.c
cvs diff -r 1.1 -kk main.c
cvs diff -D "last week" -kk main.c
cvs diff -D "4 days ago" -kk main.c
cvs diff -D "3/12/98" -kk main.c
```

Revert repository to state it was in 24 hours before. Use `-p` switch to overwrite version in current directory. This prevents CVS from applying sticky flag to new versions.

```
cvs update -D "24 hours ago" -p ncap.h > ncap.h
cvs update -D "24 hours ago" -p ncap_lex.l > ncap_lex.l
cvs update -D "24 hours ago" -p ncap_yacc.y > ncap_yacc.y
cvs update -D "24 hours ago" -p ncap_utl.c > ncap_utl.c
```

Checkout a date-stamped version of a module:

```
cvs -z3 -d zender@nco.cvs.sf.net:/cvsroot/nco co -D 20050524 -d nco-2005052
```

Compare module files from two different dates:

```
cvs diff -c --ignore-all-space -kk -D 20050519 -D 20050520
```

2.4.8 Commit

Commit changes beneath CWD. The `commit` verb accepts optional filename arguments for file-by-file (rather than entire module) commits.

```
cvs commit                # Invokes editor for log message
cvs commit -m "fixed bug" # Uses "fixed bug" for log message
cvs commit README         # Only commits README file
```

2.4.9 Tag

Tag CWD with `mdl-1_0`. Option `-c` causes `tag` to verify that files beneath CWD are not modified relative to the repository. This ensures the repository has all the information needed to exactly reproduce the CWD from the tag name in the future (with, e.g., "`cvs co -r mdl1_0 mdl`".

```
cvs tag -c mdl-1_0     # From top-level of CWD
cvs tag -c mdl-1_0 mdl # Above top-level of CWD
```

2.4.10 Rtag

"Repository" tag (`rtag`) the module with `tag_nm`. Verb `rtag`, as opposed to `tag`, operates on the repository, not the CWD. Does this only tag those portions of the repository in and beneath the CWD? (fxm). By default, `rtag` tags the most recent version of the module. This can be overridden with `-D date` option. Use (`rtag`) with caution on branches, because it will update the main repository as well (fxm).

```
cvs rtag tag_nm
```

2.4.11 Release

Release module `mdl` and delete its working directories. CVS prompts the user whether to actually delete the directory:

```
cvs release -d mdl # From level above CWD
```

2.4.12 Update

Update the CWD for `mdl` so that it reflects the latest version of the repository. One, and only one, `-j` option, "`-j vrs`", updates the CWD of `mdl` to the latest revision of the *ancestor* of `vrs`. The ancestor is the model from which the `vrs` branch split. Given two `-j` options, "`-j vrs1 -j vrs2`", the `update` command takes the differences (uses `diff`) between version `vrs1` and version `vrs2` and applies them (uses `patch`) to the CWD. The `-kk` option suppresses RCS keyword expansion, thereby minimizing the number of conflicts during an update. For this reason, it is wise to use `cvs co -kk mdl` before using `cvs update`.

```
cvs update      # Update all files in module
cvs update *    # Update currently checked-out files only
cvs update mdl  # Above top-level of CWD
cvs update -kk
```

```
cvs update -kk -d # Add directories not in working copy
cvs update -j vrs -kk
cvs update -j vrs1 -j vrs2 -kk
cvs update -kk -p params.h > foo        # Redirect file to stdout
# Replace modified file with repository file
cvs update -kk -p params.h > params.h
```

2.5 C++ Module

The C++ module `c++` was imported using "`cvs import -m "Imported sources" c++ zender c++-1_0`".

2.5.1 Checkout

Checkout the latest version of the C++ module.

```
# From NCAR, from ~
cvsbin co c++
# From home, from ~
cvs -d :pserver:zender@bearmtn.cgd.ucar.edu:/home/zender/cvs login
cvs -d :pserver:zender@bearmtn.cgd.ucar.edu:/home/zender/cvs co -kk c++
# From UCI, from ~
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/home/zender/cvs co -kk c++
```

2.5.2 Commit

Commit C++ CWD to remote repository. One of the nicest feature about CVS is that it knows about remote repositories, so commands like `commit` and `update` need no extra arguments.

```
# Update CWD at home and continue working
cvs update c++ # From home ~/
# Commit CWD at home to repository at NCAR
cvs commit c++ # From home ~/
```

2.6 NCO Module

The NCO module `nco` was imported using "`cvs import -m "Imported sources" nco zender nco-1_0`".

2.6.1 Checkout

Checkout the latest version of the NCO module.

```
# From home, from ~/nc
cvs -d :pserver:zender@bearmtn.cgd.ucar.edu:/home/zender/cvs login
cvs -d :pserver:zender@bearmtn.cgd.ucar.edu:/home/zender/cvs co nco
```

2.6.2 Update

Update a remote module (i.e., at home) to the current NCAR NCO repository.

```
cvs status
cvs -n update
cvs update      # Update all files in module
cvs update *    # Update currently checked-out files only
```

2.7 CRM Module

The CRM module `crm` was created as a branch of CCM version 3.5.22. First, we defined module `crm` by editing the `/fs/cgd/csm/models/CVS.REPOS/CVSROOT/modules` file to contain the following lines:

```
...
# CCM Column models
crm atm/ccm_crm &ccm_crm_src
ccm_crm_src -d src atm/ccm_crm_src &eul &physics &ccmlsm_share &dom &csm_sha
srchutil &control
...
```

Finally, we executed the following commands to create the CRM branch of the CCM:

```
cvs co -r ccm3_5_22 crm
cd src/crm
cvs tag -b ccm_brnch_crm
cd src/control
cvs tag -b ccm_brnch_crm [list of files in control needed by CRM]
cd src/physics
cvs tag -b ccm_brnch_crm [list of files in physics needed by CRM]
...
```

Note that we used `tag` rather than `rtag` in the above because only `tag` allows one to tag a specified subset of the files in a given module. In this case, the desired subset was the CCM files which are also required by the CRM, but no others (this excluded superfluous files, e.g., `vdiff.F`). All work on the CRM module is actually performed on this `ccm_brnch_crm` branch of the CRM, rather than the CRM main trunk This methodology, i.e., working on the branch rather than the main trunk, is also used for the Dust module component of CCM, MATCH, and MOZART described in Sections 2.8, 2.9, and 2.10 respectively.

2.7.1 Create Distribution

Release a new CRM. This process uses many CVS commands. First, make sure the `crm` module in the repository is up-to-date, so that the CWD can be exactly reproduced. Second, release the module. Third, check out the CRM with the `-kk` option. Fourth, difference the CWD with the most recent branch tag. This difference file shows the differences between the releases. Do these differences make sense? Fifth, tag the module. Sixth, execute the distribution script.

```
cd ..                    # Move above top-level of CWD
cvs commit crm           # Above top-level of CWD
cvs release -d crm       # Above top-level of CWD
cvs -d /fs/cgd/csm/models/CVS.REPOS co -r ccm_brnch_crm -kk crm
cvs diff -kk -r ccm3_6 crm > pre.diff
cvs tag -c ccm3_6_brnchT_crm2_0 crm
crm_dst.pl --dbg=1 ccm3_6_brnchT_crm2_0
```

2.7.2 Update to newer CCM version

Update the CWD to a newer CCM version. This process uses many CVS commands. First, check-out a clean copy of the CRM branch into the CWD. Second, difference the pre-merged CRM from its current CCM base. This difference file shows all the CRM-specific modifications to the current CCM base code. Third, update the CWD to the desired CCM version. Given two `-j` options, ("`-j ccm3_5_22 -j ccm3_6_0`"), the `update` command takes the differences (uses `diff`) from CCM version `ccm3_5_22` to CCM version `ccm3_6_0` and applies them (uses `patch`) to the CWD. Fourth, examine and fix the conflicts caused by this merge. Use, e.g., "`cvs status`" to locate conflicts. Make sure to compile, test, and generate new *.out files for the updated model. This ensures that any new source files will be included in `Srcfiles` and `Depends`. Also, it is important to compile with "`-DSINGLE_SOURCE_FILE`" option. This verifies that any new files merged into, and needed by, the CRM are accounted for. Unfortunately, there is no simple way to ensure that superfluous source files have not crept into the CRM. Fifth, difference the post-merged CRM from the new CCM base. This difference file shows all the CRM-specific modifications to the new CCM base code. Sixth, difference the two difference files. This file highlights conflicts caused by the merge process and, hopefully, any mistakes CVS made in performing the update. Seventh, commit the updated code to the head of the branch. Eighth, if desired, tag the branch as a new release of CRM.

```
# Above top-level of CWD
cvs -d /fs/cgd/csm/models/CVS.REPOS co -r ccm_brnch_crm -kk crm
cvs diff -kk -r ccm3_5_22 crm > pre.diff
cvs update -j ccm3_5_22 -j ccm3_6 crm
cvs status crm | grep -i conflict
cvs diff -kk -r ccm3_6 crm > post.diff
diff pre.diff post.diff > diff.diff
cvs commit crm
cvs tag -c ccm3_6_brnchT_crm2_0 crm
```

2.7.3 Checkout

Checkout the latest version of the CRM. By default, the `crm` module is placed in the `crm` directory of the CWD. A `-d` argument after the verb specifies an arbitrary directory.

```
cvs co -r ccm_brnch_crm -kk crm
cvs -d /fs/cgd/csm/models/CVS.REPOS co -r ccm_brnch_crm -kk crm
cvs -d /fs/cgd/csm/models/CVS.REPOS co -r ccm_brnch_crm -d drc crm
```

```
cvs -d :pserver:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS \
co -r ccm_brnch_crm -kk crm
```

2.7.4 Export

Export the latest version of the module `crm` from CCM branch `ccm_brnch_crm` into directory `/data/zender/crm-1.1` and prepare a compressed tarfile distribution. The `export` command strips all the CVS directories from the output. This is most useful for creating distributions for public release. Using `-kkv` instead of `-kv` would expand CVS keywords into keywords plus values.

```
/bin/rm -r -f /data/zender/crm-1.1
cvs export -kv -r ccm_brnch_crm -d /data/zender/crm-1.1 crm
cd /data/zender; gtar -cvzf crm-1.1.tar.gz ./crm-1.1
```

2.7.5 Tag

Tag CWD with `ccm3_6_brnchT_crm2_0`. Option `-c` causes `tag` to verify that files beneath CWD are not modified relative to the repository. This ensures the repository has all the information needed to exactly reproduce the CWD from the tag name in the future. Note it is wise to tag only modules which have been checked out with the `-kk` option. This prevents large numbers of trivial differences between tagged versions.

```
cvs tag -c ccm3_6_brnchT_crm2_0
```

2.7.6 Remote Checkout

Checkout the latest version of the CRM from home. The first `-d` option specifies the remote repository. The second `-d` option specifies the name of the directory to place the module in. The final argument, `crm`, is the name of the module to checkout. For unknown reasons this command erroneously places the src directories one level too high in most cases. This is apparently a CVS bug fixed in later versions of CVS.

```
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS \
co -r ccm_brnch_crm -N -d crm crm
```

Here is a workaround for the above problem:

```
mv ccmlsm_share dom physics srchutil control csm_share eul src
mv src crm
```

2.8 CCM-Dust Module

The module `ccm_dst` comprises the sub-modules `ccm` and `dst`. Thus `ccm_dst` is the complete CCM with the dust modifications. Code modifications related to dust appear in the usual files in `src` (e.g., file `physics/aphys.F`). Most of the dust physics are isolated in the new directory `src/dust`. The branch `ccm_brnch_dst` was created as a branch of CCM version 3.5.22. First, we defined module `ccm_dst` by editing the `/fs/cgd/csm/models/CVS.REPOS/CVSROOT/modules` file to contain the following lines:

```
...
# Atmospheric models
ccm             atm/ccm &ccm_src &ccm_tools
ccm_dst         atm/ccm &ccm_dust_src &ccm_tools
...
#  Sub-modules to ccm
ccm_src -d src atm/ccm_src &dynamics &physics &control &spmd &dom &som \
                &lsm &ccmlsm_share &csm_share &mathutil &srchutil
ccm_dust_src -d src atm/ccm_src &dst &dynamics &physics &control &spmd &dom
                &lsm &ccmlsm_share &csm_share &mathutil &srchutil
...
physics         atm/ccm_src_dirs/physics
dst             atm/ccm_src_dirs/dust
...
```

Finally, we executed the following commands to create the Dust branch of the CCM:

```
cvs co ccm_dst
cd ccm_dst
cvs tag -b ccm_brnch_dst
```

2.8.1 Checkout

Checkout the latest version of the CCM-Dust module. A `-d` argument after the verb specifies an arbitrary directory for the module.

```
cd /fs/cgd/data0/zender
cvsbin -d /fs/cgd/csm/models/CVS.REPOS co -r ccm_brnch_dst -kk ccm_dst
# From home, from ~/fsh/dst
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS \
co -r ccm_brnch_dst -kk ccm_dst
cvs -d :pserver:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS \
co -r ccm_brnch_dst -kk ccm_dst
# From UCI
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS co -r
# CCM Box model only
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS co -r
# Standard CCM, from UCI
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS co -k
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS co -r
# Source directory only
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS co -r
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS co -k
# Then when updating parent, use cvs update -l -kk for local update only
# Marianna's dust CLM
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS co -r
```

```
# Natalie's dust CAM/CLM
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS co -r
# Head of CAM
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS co -r
# SNICAR
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS rtag
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/models/CVS.REPOS co -k
```

2.8.2 Query

Show which files in CWD have changed relative to the most recent committed version in the Dust repository. Then show precisely what has changed for file `dstmbl.F`.

```
cvs -nq update -kk
```

The result of `cvs -n update` is a list of files which differ from the repository. Each file is prefixed by a code. The most common codes are `C` (for conflicts), `M` (local modifications newer than repository), and `U` (local file needs updating). Conflicts

```
/home/zender/f: cvs -n update
cvsbin update: Updating .
C Makefile
M csz_f77.F
U sng.F
```

The precise differences relative to the repository are shown with

```
cvs diff -kk
```

Show which files in CWD have changed relative to version `mdl-1_0` in the CCM repository. Then show precisely what has changed for file `Makefile`.

```
cvs -nq update -r mdl-1_0 -kk
cvs diff -r mdl-1_0 -kk Makefile
```

2.8.3 Release

Release the current Dust module and check out a new working copy.

```
# Above top-level of CWD
cvs -d /fs/cgd/csm/models/CVS.REPOS release -d ccm_dst
cvs -d /fs/cgd/csm/models/CVS.REPOS co -r ccm_brnch_dst -kk ccm_dst
```

2.8.4 Tag

Note that tagging in the CCM repository invokes error checking modules and, by default, launches windows which request one line summaries on the local machine. Attempting to tag from behind a firewall or proxy can thus result in failure because the windows are unable to open on the requested console. In this case, the tagging must be done from goldhill.cgd.ucar.edu, and the user's `DISPLAY` must be set to `NONE`

```
export DISPLAY=NONE
```

This problem does not arise when tagging from an unfiltered IP address.

Tag CWD with `ccm3_6_brnchT_dst1_2`. Option `-c` causes `tag` to verify that files beneath CWD are not modified relative to the repository. This ensures the repository has all the information needed to exactly reproduce the CWD from the tag name in the future.

```
cd ccm_dst               # Move to top-level of CWD
cvs tag -c ccm3_6_brnchT_dst1_2
```

2.8.5 Update

Update the CWD to a newer CCM version.

```
# Above top-level of CWD
cvs -d /fs/cgd/csm/models/CVS.REPOS co -r ccm_brnch_dst -kk ccm_dst
cvs -d /fs/cgd/csm/models/CVS.REPOS diff -kk -r ccm3_5_22 ccm_dst > pre.dif
cvs -d /fs/cgd/csm/models/CVS.REPOS update -j ccm3_5_22 -j ccm3_6 ccm_dst
cvs -d /fs/cgd/csm/models/CVS.REPOS status ccm_dst | grep -i conflict
cvs -d /fs/cgd/csm/models/CVS.REPOS diff -kk -r ccm3_6 ccm_dst > post.diff
diff pre.diff post.diff > diff.diff
cvs commit ccm_dst
cvs tag -c ccm3_6_brnchT_dst1_0 ccm_dst
```

2.9 MATCH-Dust Module

The module `match_dst` comprises the sub-modules `match` and `dst`. Thus `match_dst` is the complete MATCH with the dust modifications. Code modifications related to dust appear in the usual files in `src` (e.g., file `physlic.F`). Most of the dust physics are isolated in new files in the `dst` directory, e.g., `dstmbl.F`. The branch `match_dst-0` was created as a branch of MATCH Spitfire version 3.2.9. First, we defined module `match_dst` by editing the `/fs/cgd/csm/people/eaton/C` file to contain the following lines:

```
match           match
matchSrc        -d src match/src
matchReaders    -d readers match/readers
...
dst             dust
match_dst       match &dst
```

The `dust` source directory in the MATCH repository is actually a symbolic link to the dust source in the CSM repository:

```
cd /fs/cgd/csm/people/eaton/CVS
ln -s /fs/cgd/csm/models/CVS.REPOS/atm/ccm_src_dirs/dust .
```

Next, we added a MATCH tag to the CCM branch of the Dust module.

```
cvs -d /fs/cgd/csm/models/CVS.REPOS co -kk -r ccm_brnch_dst dst
cd dst
cvs tag -c MATCH_SPITFIRE-3_2_9
```

This enabled us to check out both MATCH and Dust at the same time in preparation for the next step, creating the MATCH-Dust branch. Note that in order to apply MATCH tags to files in the CCM repository, we had to disconnect the CCM-Dust module from the default CCM tag syntax checking routines. Finally, we created and tagged the MATCH-Dust branch.

```
cvs -d /fs/cgd/csm/people/eaton/CVS rtag -b -r MATCH_SPITFIRE-3_2_9 \
match_brnch_dst match_dst
```

In the preceding command, `MATCH_SPITFIRE-3_2_9` is the pre-existing tag associated with all files that were placed in the new `match_brnch_dst` branch of the `match_dst` module. Originally, we left the `MATCH_SPITFIRE-3_2_9` tag on the files in the `dst` directory, but this caused problems when merging changes to the main trunk of the MATCH into MATCH-Dust. The tag caused CVS to think that the Dust-specific files were parts of `MATCH_SPITFIRE-3_2_9`, and thus CVS tried to remove them when updating to a later version on the main trunk. Thus, we later removed the `MATCH_SPITFIRE-3_2_9` tag from all the files in the `dst` directory.

2.9.1 Changes to standard MATCH repository

Once these commands were completed, we slightly altered the architecture of the MATCH-Dust repository to facilitate CCM-style compiling methods (i.e., compile from list of directories rather than list of individual files plus directories). `match_brnch_dst` was created from the MATCH 3.2 repository, and these commands were used to modify the standard MATCH repository:

```
# Routine to read CCM dynamics data conflicts with NCEP reader routine.
# Store it in parallel, non-conflicting, directory:
cd /fs/cgd/data0/zender/match_dst/readers
mkdir ccm
cvs add ccm
cd ccm
mv /fs/cgd/data0/zender/match_dst/src/dyninp.F .
cvs remove /fs/cgd/data0/zender/match_dst/src/dyninp.F
cvs add dyninp.F

/bin/rm extoro.F setzen.F # Remove relics from Mark Lawrence
cvs remove extoro.F setzen.F
/bin/rm cloud_dum.F # Routines superceded by prognostic cloud water
cvs remove cloud_dum.F

cd ..
cvs commit -m "Rearranged files to work with MATCH-Dust Makefile"
```

To create a dust branch from the standard MATCH 4.X repository, the following changes to the standard directory structure would be required:

```
# Routine to read CCM dynamics data conflicts with NCEP reader routine.
# Store it in parallel, non-conflicting, directory:
cd /fs/cgd/data0/zender/match_dst/readers
mkdir ccm
cvs add ccm
cd ccm
mv /fs/cgd/data0/zender/match_dst/src/dyninp.F .
cvs remove /fs/cgd/data0/zender/match_dst/src/dyninp.F
cvs add dyninp.F

# Remove routines in libncaru.a and libmss.a on NCAR computers
/bin/rm setzen.F # Remove relics from Mark Lawrence
cvs remove setzen.F

cd ..
cvs commit -m "Rearranged files to work with MATCH-Dust Makefile"
```

2.9.2 Checkout

Checkout the latest version of the MATCH-Dust module. A `-d` argument after the verb specifies an arbitrary directory for the module.

```
# Get MATCH-Dust on Dataproc
cd /fs/cgd/data0/zender
cvs -d /fs/cgd/csm/people/eaton/CVS co -r match_brnch_dst -kk match_dst
# Get MATCH-Dust in CGD
cvs -d :pserver:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/people/eaton/CVS \
co -r match_brnch_dst -kk match_dst
# Get MATCH-Dust at UCI
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/people/eaton/CVS co -r
# Get Dust box model at UCI (i.e., checkout the dst directory, name it aer)
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/people/eaton/CVS co -r
# Get Assimilation model at UCI
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/people/eaton/CVS co -r
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/people/eaton/CVS co -r
# Create Pat's branch
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/people/eaton/CVS rtag -
# Checkout Pat's branch (``old'' model)
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/people/eaton/CVS co -r
# Create Alf's sandblasting branch
cvs tag -b -r match-4_0_beta2-dst-1_1_1 match_brnch_dst_sbl match_dst
# Checkout Alf's sandblasting branch
cvs -d :ext:zender@goldhill.cgd.ucar.edu:/fs/cgd/csm/people/eaton/CVS co -r
# Get Dust box model from ESMF (i.e., model directory is named dead)
cvs -d :ext:zender@esmf.ess.uci.edu:/u/zender/cvs co -kk -r match_brnch_dst
```

2.9.3 Tag

Tag CWD with `match-3_2_10-dst-1_3`. Option `-c` causes `tag` to verify that files beneath CWD are not modified relative to the repository. This ensures the repository has all the information needed to exactly reproduce the CWD from the tag name in the future. Be sure this version compiles before tagging it.

```
cd match_dst             # Move to top-level of CWD
cvs tag -c match-3_2_10-dst-1_3
```

2.9.4 Merge CCM changes into CCM-Dust

Merge changes to main CCM trunk into the CCM-Dust branch. Merging main trunk changes into CCM-Dust should never change the `dst` directory itself, but should alter the contents of `src/physics`, `src/control`, etc. This example merges the changes from CCM version `ccm3_6` to CCM version `ccm3_6_6` into CCM-Dust version `ccm3_6_brnchT_dst-0_9_6`, which is then tagged as `ccm3_6_6_brnchT_dst-0_9_6`.

```
cd /fs/cgd/data0/zender/ccm_dst
cvs diff -kk -r ccm3_6 > pre.diff
cvs update -kk -j ccm3_6 -j ccm3_6_6
cvs status | grep -i conflict
cvs diff -kk -r ccm3_6_6 > post.diff
diff pre.diff post.diff > diff.diff
make ccm
cvs commit
cvs tag -c ccm3_6_6_brnchT_dst-0_9_6
```

2.9.5 Merge MATCH changes into MATCH-Dust

Merge changes to main MATCH trunk into the MATCH-Dust branch. Merging main trunk changes into MATCH-Dust should never change the `dst` directory itself, but should alter the contents of `src`, `readers`, etc. This example merges the changes from MATCH version `match3_2_10_scyc1_0` to MATCH version `match3_3_20` into MATCH-Dust version `match-3_2_10-dst-1_4a`, which is then tagged as `match-3_3_20-dst-1_4a`.

```
cd /fs/cgd/data0/zender/match_dst
cvs diff -kk -r match3_2_10_scyc1_0 > pre.diff
cvs update -kk -j match3_2_10_scyc1_0 -j match3_3_20
cvs status | grep -i conflict
cvs diff -kk -r match3_3_20 > post.diff
diff pre.diff post.diff > diff.diff
make match
cvs commit
cvs tag -c match-3_3_20-dst-1_4a
```

2.9.6 Merge MATCH-Dust changes into CCM-Dust

Merge changes to the MATCH-Dust branch into the CCM-Dust branch. A simple "`cvs update`" does not work because the changes are on different branches. This example merges the changes from MATCH-Dust version `match-3_2_9-dst-1_2` to MATCH-Dust version `match-3_2_10-dst-1_4` into CCM-Dust version `ccm3_6_brnchT_dst1_2`, which is then tagged as `ccm3_6_brnchT_dst1_4`.

```
cd /fs/cgd/data0/zender/ccm_dst/src/dst
cvs diff -kk -r ccm3_6_brnchT_dst1_2 > pre.diff
cvs update -kk -j match-3_2_9-dst-1_2 -j match-3_2_10-dst-1_4
cvs status | grep -i conflict
cvs diff -kk -r match-3_2_10-dst-1_4 > post.diff
diff pre.diff post.diff > diff.diff
make ccm
cvs commit
cvs tag -c ccm3_6_brnchT_dst1_4
```

2.9.7 Merge CCM-Dust changes into MATCH-Dust

Merge changes to the CCM-Dust branch into the MATCH-Dust branch. A simple "`cvs update`" does not work because the changes are on different branches. This example merges the changes from CCM-Dust version `match-3_2_9-dst-1_0` to CCM-Dust version `ccm3_6_brnchT_dst1_2` into MATCH-Dust version `match-3_2_9-dst-1_0`, which should then be tagged as `match-3_2_9-dst-1`

```
cd /fs/cgd/data0/zender/match_dst/dst
cvs diff -kk -r match-3_2_9-dst-1_0 > pre.diff
cvs update -kk -j match-3_2_9-dst-1_0 -j ccm3_6_brnchT_dst1_2
cvs update -p -j match-3_2_9-dst-1_0 params.h > params.h
cvs status | grep -i conflict
cvs diff -kk -r ccm3_6_brnchT_dst1_2 > post.diff
diff pre.diff post.diff > diff.diff
cvs commit
make match
cvs tag -c match-3_2_9-dst-1_2
```

2.10 MOZART-Dust Module

The module `mozart_dst` comprises the sub-modules `mozart` and `dst`. Thus `mozart_dst` is the complete MOZART with the dust modifications. The MOZART Dust implementation was created well after MATCH-Dust, and uses most of the same directory structure. Thus, the CVS commands for manipulating the MOZART-Dust model are directly analogous to those described in Section 2.9, with the following exceptions.

2.10.1 Field names

By default, MOZART archives both instantaneous values of constituents mixing ratios. These are given the names assigned in the code, e.g., `DSTQ01`. MOZART can also be told to archive time-

averaged values of these constituent fields. Time-averaged constituent fields are assigned names based on there constituent index in the model, e.g., `TRA01`.

3 Subversion Overview

3.1 Modules

To create a Subversion repository, use

```
svnadmin create /home/zender/svn
export SVNROOT="svn+ssh://dust.ess.uci.edu/data/home/${USER}/svn/trunk"
```

3.2 Generic Modules

3.2.1 Import an existing directory

The `cvs2svn` command converts CVS repositories to Subversion repositories.

```
# Dry run do not actually commit anything.
cvs2svn -v --dry-run -s /home/zender/nco esmf.ess.uci.edu:/u/zender/cvs
cvs2svn -v --dry-run -s /home/zender/svn /home/zender/cvs
cvs2svn -v --dry-run -s /var/www/html/svn esmf.ess.uci.edu:/u/zender/cvs
# Commands actually used
cvs2svn -v --existing-svnrepos --force-tag=map-2_1_2 \
        -s /home/zender/svn /home/zender/cvs
# Valid Subversion keywords are:
Date Revision Author HeadURL Id
By default cvs2svn expands CVS keywords Name, Revision, Header
prior to importing to Subversion.
cvs2svn correctly transitions CVS keywords Date, Id to Subversion.
# Keyword expansion appears to be automatic: How to turn off?
cvs2svn --keywords-off
# Change mode (chmod) of local file to executable
cd ~/aca;svn propset svn:executable ON htrn2nc.pl aca.pl
cd ~/aca;svn propset svn:executable ON htrn2nb.sh htrn2nc.sh aca.sh etags.sl
# Find all files changed since 20081118
cvs diff -N -c -D 20081118 | grep "Index:" > ~/foo.new
# Backup ESMF CVS
cd /home/zender
tar cvzf ./cvs.tar.gz ./cvs
scp cvs.tar.gz dust.ess.uci.edu:
# Production conversion on PBS
/bin/rm -r -f ~/svn
cvs2svn -v -s /home/zender/svn /home/zender/cvs > ~/cvs2svn.txt
# keywords-off does not help me email from DLW 20081125
```

```
# cvs2svn --keywords-off -v -s /home/zender/svn /home/zender/cvs > ~/cvs2svn
# http://www.cvsnt.org/manual/html/Substitution-modes.html
# To switch off keyword substitution for all files in a subtree:
cvs update -k +o
cvs commit -m "Change substitution mode"
```

Import new source directories:

```
svn help import
cd mdl
svn import -m "Imported sources" ~/mdl ${SVNROOT}/mdl
cd ..
mv mdl mdl.bck
svn co ${SVNROOT}/mdl
ls -R mdl
/bin/rm -r mdl.bck
```

Convert my CVS repositories from ESMF to PBS 20090205:

```
rps_usr='zender/cvs'
#rps_srv_old='dust'
#rps_srv_new='esmf'
rps_srv_old='esmf'
rps_srv_new='dust'
export CVS_RSH='ssh'    # Needed for ssh access to NCAR CGD CVS
export CVSROOT=":ext:${USER}@${rps_srv_new}.ess.uci.edu:${HOME}/cvs"
echo ":ext:${rps_srv_new}.ess.uci.edu:/home/zender/cvs" > ~/cvs_zender.txt
locate CVS/Root | grep zender > ~/cvs_Root.txt
for fl in `cat cvs_Root.txt`; do
rps_crr=`cat ${fl}`
case "${rps_crr}" in
    *${rps_srv}*${rps_usr}* )
    printf "${fl} points to a ${rps_usr} repository on ${rps_srv_old}, will
    /bin/cp -f ~/cvs_zender.txt ${fl}
    ;; # endif match*
    * )
    printf "${fl} does not point to a ${rps_usr} repository on ${rps_srv_old
    ;; # endif default
esac # endcase ${rps_crr}
done
# Server instructions here were not helpful:
# http://sanatio.blogspot.com/2005/12/cvs-server-on-ubuntu.html
# Instead, had to manually create lock directories on server with
sudo mkdir /var/locks
sudo mkdir /var/locks/cvs
sudo chmod 777 /var/locks
```

```
sudo chmod 777 /var/locks/cvs
# 20150216: Sourceforge reported lock errors
# https://sourceforge.net/p/forge/site-support/9688
# Took a week to fix, no thanks to SF, finally figured this out:
rsync -av nco.cvs.sourceforge.net::cvsroot/nco/* . # Backup CVS repo
ssh -t zender,nco@shell.sourceforge.net create # Create login shell
sf-help --web # Print location of hidden directories
cd /home/project-web/nco # Apparently this is my repo
adminrepo --unlock cvs # This gets rid of the dreaded locks
adminrepo --save # This did not actually seem to help
```

Check out modules:

```
svn checkout file:///home/zender/svn/trunk/f ~/foo
svn checkout file:///home/zender/svn/trunk/mdl ~/mdl
```

Check out date-stamped or version-stamped module into new directory foo

```
svn checkout --revision {2009-11-11} $SVNROOT/crr ~/foo
svn checkout --revision {2013-12-02} file:///home/zender/svn/trunk/nco ~/foo
```

Export date-stamped or version-stamped file into new file foo

```
svn export --revision {2009-11-11} $SVNROOT/crr/crr.txt ~/foo
```

Examine status or working directory

```
svn status
```

Commit changes in working directory

```
svn
```

List modules contained in repository

```
svn ls svn+ssh://zender@dust.ess.uci.edu/home/zender/svn/trunk
svn checkout file:///home/zender/svn/trunk/dot ~/dot
```

Tagging

```
svn copy svn+ssh://zender@dust.ess.uci.edu/home/zender/svn/trunk/f \
svn+ssh://zender@dust.ess.uci.edu/home/zender/svn/tags/f/2.0.3 \
-m "Tagging the 2.0.3 release of the project"
```

Equivalence between `svn` and `svn+ssh` methods:

```
# Using Subversion only full path not necessary, but svnserve must run on se
svn checkout svn://dust.ess.uci.edu/svn/trunk/f
# Using Subversion + SSH, full path is necessary, but svnserve not used:
svn checkout svn+ssh://dust.ess.uci.edu/home/zender/svn/trunk/f
```

Converting CVS keywords to Subversion properties: First, find the files with keywords, e.g.,

```
egrep -rl '\$(Author|Date|Header|Name|Revision|Source|State|Id)' *
```

Determine the appropriate properties to set by searching Subversion's hidden .svn directory, e.g.,

```
# Convert $Id$ to $Id$ keyword
perl -pi -e 's/\$Header\$/\$Id\$/g;' *
perl -pi -e 's/\$Id$/\$Id\$/g;' *
egrep -rl '\$Id: ' * | grep -v /.svn/ | xargs svn propset svn:keywords Id
svn commit
```

Resolving conflicts: After editing to resolve a conflict, one may still receive errors on commits attempts like

```
zender@greenplanet:~/ess_gcm$ svn commit -m ""
svn: Commit failed (details follow):
svn: Aborting commit: '/state/partition1/home/zender/ess_gcm/cam_gcm.sh' re
```

The solution is to explicitly tell Subversion that the conflict has been resolved

```
zender@greenplanet:~/ess_gcm$ svn resolved cam_gcm.sh
Resolved conflicted state of 'cam_gcm.sh'
zender@greenplanet:~/ess_gcm$ svn commit -m "Resolved funky conflict"
Sending        cam_gcm.sh
Transmitting file data .
Committed revision 14055.
```

Renaming files:

```
svn mv foo bar
```

Create modules:

```
cd
svn import mdl ${SVNROOT}/mdl -m "Imported Sources"
mv mdl mdl.bck
svn co ${SVNROOT}/mdl
ls -R mdl
/bin/rm -r mdl.bck
```

4 Git Overview

ACME favors Git, and ACME-flavored development procedures for Git are described here.

4.1 Cloning

It is much more convenient to clone repositories with SSH than with HTTPS because SSH replaces the cumbersome key-management/wallet apps that vary between OS's. To clone a remote repository using SSH protocol, first install an SSH key from the development machine to the remote server (e.g., GitHub), by following these instructions .

```
git clone git@github.com:ACME-Climate/DiagnosticsWorkflow
```

To clone a remote repository using HTTPS protocol, use

```
git clone https://github.com/ACME-Climate/DiagnosticsWorkflow
```

Note that OLCF does not support HTTPS protocol. On OLCF machines, one must use clone Git repositories with the SSH protocol instead. To change the access protocol for a checked-out repository from HTTPS to SSH use

```
git remote set-url origin git@github.com:username/repository.git
git remote set-url origin git@github.com:czender/prv.git
```

4.2 Modules

4.3 Creating

```
cd ~/diwg
# Transform current directory into Git repository. Create .git subdirectory
git init
# Push new local Git repository to GitHub
git remote add origin https://github.com/czender/diwg.git
git push origin --all
git push origin --tags
git push --set-upstream origin master
```

4.4 Merging

A Git `pull` is basically a merge. If one pulls the remote master onto a local repository which contains changes, Git complains that the pull will overwrite the changes, then quits. To force the overwrite of any local changes, one uses `fetch` to update the local metadata, then `reset` to point the local repository to the remote head.

```
# Fetch from default remote, origin
git fetch
# Reset current branch (master) to origin's master
git reset --hard origin/master
```

This procedure is often necessary with NCO on older machines that automatically generate different .cpp and .hpp files from newer machines.

Merging changes in branches to the master

```
# Merging NCO branches
# This method pushes everthing from devel to master
# Works as intended when devel and master have same files
cd ~/nco
git checkout master
git pull origin master
git merge devel
git push origin master

# Merging ACME branches
# Above method does not work as intended because devel and master have (some
# Assume we are working on devel and make changes we want to incorporate int
# First, commit the changes to devel (so they are not lost when checking out
cd ~/PreAndPostProcessingScripts/generate_climatologies
git commit -am "Changes to devel"
# Then follow this recipe
git fetch
git checkout master
git pull
git checkout devel climo_nco.sh climo_ann.sh
#git status
#git diff --cached
git commit -am "message"
git push origin master
git checkout devel
```

Merging changes in master out to the branches

```
# Merging occurs locally (no pull request necessary)
# Push to remote when satisfied with results
# Graphical explanation at:
# http://stackoverflow.com/questions/3876977/update-git-branches-from-master
git checkout rgr
git merge master # Retains true history, best solution
# Any CONFLICTS are identified by <<<< HEAD ==== master >>>> syntax and must
# Branch will be called HEAD
# Unclear what to do with ChangeLog (see graphic) above:
# Edit to keep monotonic in time or let-alone preserve true sequence of merg
git commit -a -m ``Merged changes in master into rgr branch''
# ...or...
git checkout rgr
git rebase master # Retains false history, inferior solution
git commit -a -m ``Rebased rgr branch to master''
```

4.5 Workflow

Use this workflow on projects like Terraref that want one pull-request (PR) per feature to merge to master.

```
# Create new branch
git checkout -b newfeature
# Make local changes
git commit -am 'foo'
# Push local changes
git push origin newfeature
# Feature based workflow (https://www.atlassian.com/git/tutorials/comparing-
git push -u origin newfeature
# Delete local branch once it is merged upstream
git branch -d newfeature
```

4.6 Renaming

Rename local and remote branches

```
git branch -m old_branch new_branch       # Rename branch locally
git push origin :old_branch               # Delete old branch
git push --set-upstream origin new_branch # Push new branch, set local branc
```

4.7 Forensics

Obtain dated or tagged versions of code

```
git
```