CCSM4 scripts update

CCSM4 Scripts Update

please contact tcraig to propose changes
create_newcase

- Ability to set casename, mach, res, compset on command line
- Ability to run create_newcase from another dir and create a case in a different dir location
- Ability to "set" env_conf, env_run, etc options on the command line
- Ability to skip_rundb
- Abillty to clone
- Freeze components after create_newcase

create_test

- Ability to set full testname on command line, remove ability to set individual test/mach/res/compset (what about handling delimeters in the full testname)
- Ability to generate and compare to baselines
- Ability to run create_test from another dir and create a case in a different dir location
- Ability to set testid manually or have something automatically generated
- ? Ability to turn on debug and other options (how)
- ? Ability to set pes in a test manually on create_test step (is there another way to do it?)
- ? Ability to set env_conf, env_run, etc options on the command line (new)
- Ability to set clean-up flag
  create_suite
- Ability to set an input_list of tests
- Ability to set/override machine name on the command line
- Ability to run create_suite from another dir and create cases in a different dir location
- Ability to generate and compare to baselines
- Ability to set clean-up flag
- Remove ability to run the baseline suite at the same time
- Ability to set testid manually or have something automatically generated
  configure
- Ability to setup Build/Run scripts for a specific machine
- Ability to add mach, do we support two machine sets of scripts at the same time, how about cache files?
- Ability to clean/reset (simplify/clarify cleanall, reallycleanall, cleanmach, rmmach)
  env_* files
- Ability to easily modify setup via env variables
- ? Should there be one file, files associated with function (pes, mach, namelist), or files associated with setup mode (configure, build, prestage, namelist/run)
  Other
- ? Support a flag for prestage or no prestage, "prestage" data management requirements...
- Cleanly separate case creation, configure, configure pes, prestage, build, run
- (namelist setup). Should there be prestage, build, and run (namelist setup) for each component or should each component have a single file with multiple phases or should there be a single prestage, build, and run scripts with multiple components stuff in each.
- ? Ability to run the same case on multiple machines from the same directory
- ? Ability to change pes without having to call configure again. can this be done. do we need a separate tool. is this desirable. has impacts on batch submission, and build for some models. implement a configure pes phase
- ? Interaction between ccsm scripts and component "template" scripts, minimize dependencies, what is the hierarchy, etc
- ? Improve reconfigure and rebuild logic. Trap situations better. ccsm_listcache, .cache dir, cache files, etc.
- Standardize true/false TRUE/FALSE as well as shell vs env variable names
- Want relatively resolved namelist for all components accessible to users consistently
- Need to support Case SourceMods
- Support hierarchy of setting default pe layouts cleanly; machine, compset, resolution, high/med/lo pe count, run length
- ? Auto setting of run length based on resolution, pes, test, etc
- Ability to archive case into a database
- Create copies of actual namelist files in case directory
- ? Consistent build strategy, see bugs
- Consistent machine name conventions, linux vs Linux vs LINUX, names like X.ibm.bluevista -> X.bluevista
- ? Should the run script call the build script, what about the prestage script? is the prestage script part of the run script or build script or separate
- Fix build hierarchy, it's not working well now due to dependencies and naming convention
- Need to make sure things that are supported at the create_newcase level are also supported through create_test and create_suite
- Verify and improve "clean" of test runs
- The CASE script do NOT have to be run-able from another directory . for instance, my_case/configure -cleanall does NOT have to work. or $CASEROOT/my_case.build does NOT have to work. this would require a significant increase in complexity in script implementation.

```
==================================
```

What to do

Now:

- verify "local" compset capability in create_newcase
- modify scripts so they assume components are fixed from create_newcase
- implement 3 cache level checks (comps, conf, pes) using new env/cache design (still to be determined)
- add configure pes feature

- cleanup the configure clean options
- add baselineroot to env_mach files and provide override option on create_suite/test
  Next:
- cleanup naming like ibm, AIX, linux, Linux, LINUX, linux-pgi, etc
- fix build hierarchy and naming convention. do we want to go to separate Macros files for each machine without a "generic" one.
  Background:
- cleanup true/TRUE standard
  Later:
- prestage feature

```
========================
```

Design thoughts on scripts "cache" issues

- needs to support multiple levels in a relatively straight-forward way
  - create cache file/files at appropriate level
  - check cache file/files at appropriate level
  - clean cache file/files at appropriate level
  - update cache file/files at appropriate level
- needs to be clear to users what is frozen and when.

from the user perspective, we could implement this as <font color=red>(A) different env files for pre-configure, pre-configure-pes, pre-build, and pre-run

- positive: clear, file content clean (no attributes). documentation and implementation are coupled. cache files would probably just be copies of env files (easy to implement, low risk).
- negative: env variables could change files periodically which might be confusing to users. closely associated variables could end up in different files. forces env file convention. would lead to splitting env_mach file into other env files.
  this would look like

```
env_comps:
  setenv COMP_ATM       UNSET          # cam  | datm | xatm | satm
env_conf:
  setenv CAM_CHEM       none           # none | trop_mozart |
env_pes:
  setenv NTASKS_ATM 16
env_build:
  setenv DEBUG         FALSE           # TRUE, FALSE
env_run:
  setenv STOP_OPTION   ndays           # calendar_alarm; ndays, nmonths, daily
```

(B) add "attribute" associated with subset of env variables

- positive: can change attribute easily. can have as many env files as wanted/needed organized in any way (by component, by phase of scripts, etc). documentation and implementation are coupled. attributes can be much more descriptive than B due to extra space.
- negative: requires additional attribute for variables, cluters env files a bit. have to define the attribute tokens clearly but want them to be compact, concern about extensiblity in the future depending on attribute naming convention. requires file parsing to set cache files (more difficult implementation and higher risk).
  this would look like

```
env_??
  setenv COMP_ATM       UNSET   #FRZ_COMP       # cam  | datm | xatm | satm
  setenv CAM_CHEM       none    #FRZ_CONF       # none | trop_mozart |
  setenv NTASKS_ATM 16          #FRZ_PES
  setenv DEBUG         FALSE    #FRZ_BLD       # TRUE, FALSE
  setenv STOP_OPTION   ndays                    # calendar_alarm; ndays, nmonths, daily
```

where #FRZ_* is the attribute associated with freezing/setting the cache level. what should those special tokens/names be, need to be able to change levels and add levels easily if needed.

(B1) add "attribute" associated with subset of env variables via independent lines

- positive: can change attribute easily. can have as many env files as wanted/needed organized in any way (by component, by phase of scripts, etc). documentation and implementation are coupled. similar approach as xml.
- negative: env files will be quite a bit bigger. more complex parsing due to multiple lines per variable. requires refactor of "sed" tool which will require two passes per file. for parsing sanity, format will need to be well defined and error checking will always be incomplete. requires additional attribute for variables. have to define the attribute tokens clearly but want them to be compact, concern about extensiblity in the future depending on attribute naming convention.

```
env_??
  setenv COMP_ATM    UNSET
  # COMP_ATM cache_level:     frz_comp
  # COMP_ATM valid_values:    cam  | datm | xatm | satm
  # COMP_ATM description :    this could be a place holder for a one line description
  setenv CAM_CHEM    none
  # CAM_CHEM cache_level:     frz_conf
  # CAM_CHEM valid_values:    none | trop_mozart | waccm_mozart |
  # CAM_CHEM description:
  setenv NTASKS_ATM  16
  # NTASKS_ATM cache_level:     frz_pes
  # NTASKS_ATM valid_values:    na
  # NTASKS_ATM description:
  setenv DEBUG       FALSE
  # DEBUG cache_level      frz_bld
  # DEBUG valid_values:    true | false
  # DEBUG description:
  setenv STOP_OPTION ndays
  # STOP_OPTION cache_level      none
  # STOP_OPTION valid_values:    na
  # STOP_OPTION description      calendar_alarm; ndays, nmonths, daily,
```

(C) have cache-ing managed entirely by a Tool. have scripts comments and user guides inform users what variables are "frozen" at each step. (this is what we're doing now, although documentation is poor)

- positive: cache tool is more independent. env files are independent. clean-ish env files. can have as many env files as wanted/needed organized in any way (by component, by phase of scripts, etc). with good documentation, the implementation should be clearer to users than using short-ish attributes. cache files set by writing env variables (easy to implement, low risk).
- negative: documentation needs to remain up-to-date. decouples documentation from implementation.
  this would look like

```
env_??
  # these variables should never be changed:
  setenv COMP_ATM       UNSET   # cam  | datm | xatm | satm
  # these variables should not be changed after configure:
  setenv CAM_CHEM       none    # none | trop_mozart |
  # these variables should not be changed after configure pes:
  setenv NTASKS_ATM 16
  # these variables should not be changed after building the model:
  setenv DEBUG         FALSE    # TRUE, FALSE
  # these variables can be changed anytime:
  setenv STOP_OPTION   ndays    # calendar_alarm; ndays, nmonths, daily
```

What if we worked on some auto-documentation tool based on the ccsm_listcache file. we might even be able to get the env_ file auto documented based on the current ccsm_listcache.

Some env/cache details

- implement a general env "source" script in ccsm_utils/Tools that would be used to add or delete env file easily without having to change all the scripts. use "-f" if test for individual files.
- still need a tool to create/check cache files. need a "levels" feature