



TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU



TEXAS

The University of Texas at Austin

Settarg

Robert McLay

Dec. 7, 2021

Settarg



- ▶ What is settarg and how did it get started?
- ▶ What are the goals of settarg?
- ▶ What does settarg actually do?
- ▶ Examples
- ▶ Personalizing settarg: user, projects
- ▶ Conclusions

What is settarg?

- ▶ The settarg module connects the build system to:
 - ▶ What modules are loaded: compiler, mpi, ...
 - ▶ Type of build: debug/optimize/...
 - ▶ Machine Arch: x86_64, ppc64le, arm64, ...
- ▶ Provides environment variables to connect to the build system
- ▶ Makes it easy to switch between different builds
- ▶ Integrated with Lmod and is installed with it.

Story: What led to settarg

- ▶ When processor speeds were measured in MHz and not GHz
- ▶ Worked in a lab that had incompatible architectures with a shared home file system.
- ▶ Wanted to build programs w/o doing a `make clean` in-between.
- ▶ How about different optimization levels? (dbg, opt, mdbg?)
- ▶ Different compilers or MPI stacks?
- ▶ Settarg became part of Lmod (because I can't remember to do two things at once!?)

Where the settarg name came from

- ▶ Originally wanted \$TARGET
- ▶ But many other projects (PETSc, etc) already use that
- ▶ \$TARGET shorten to \$TARG
- ▶ This tool was designed to set \$TARG ⇒ [settarg](#)

Goals of settarg

- ▶ Make switching between DBG/OPT builds easier.
- ▶ Make switching between compiler/mpi and cmplr/mpi versions easier
- ▶ Integrate with Build process via Env. Vars.
- ▶ Integrate with \$PATH
- ▶ Optionally report status in titlebar
- ▶ Integrate with Lmod!

The 4 things that settarg does

- ▶ Reads the state of the loaded modules
- ▶ Build \$TARG variables
- ▶ Change \$PATH with new \$TARG
- ▶ Optionally changes the titlebar

Reasons to use settarg

- ▶ Switching compiler can be helpful.
- ▶ C++ error msgs are confusing,
- ▶ Different compilers might make better sense.
- ▶ Debugging with gcc; Use intel for performance
- ▶ Comparing two different version of a compiler with your application.

Typical \$TARG variables in dbg state

- ▶ TARG: OBJ/_x86_64_dbg_gcc-9.3.0_mpich-3.3.2
- ▶ TARG_BUILD_SCENARIO: dbg
- ▶ TARG_COMPILER_FAMILY: gcc
- ▶ TARG_MPI_FAMILY: mpich
- ▶ TARG_COMPILER: gcc-9.3.0
- ▶ TARG_MPI: mpich-3.3.2

Typical \$TARG variables in opt state

- ▶ TARG: OBJ/_x86_64_opt_gcc-9.3.0_mpich-3.3.2
- ▶ TARG_BUILD_SCENARIO: opt
- ▶ TARG_COMPILER_FAMILY: gcc
- ▶ TARG_MPI_FAMILY: mpich
- ▶ TARG_COMPILER: gcc-9.3.0
- ▶ TARG_MPI: mpich-3.3.2

\$PATH and \$TARG

- ▶ settarg inserts \$TARG into \$PATH.
- ▶ dbg \Rightarrow PATH=OBJ/_x86_64_dbg:~/bin:./usr/local/bin:/bin
- ▶ opt \Rightarrow PATH=OBJ/_x86_64_opt:~/bin:./usr/local/bin:/bin

\$PATH searching can be dynamic!

- ▶ Normally shells build a table of all exec's in path
- ▶ rehash can be required for new exec's
- ▶ But relative paths are evaluated *everytime*.

The settarg module defines the following commands

- ▶ `dbg` \Rightarrow debug
- ▶ `opt` \Rightarrow optimize
- ▶ `mdbg` \Rightarrow max debug
- ▶ `empty` \Rightarrow no build scenario
- ▶ `cdt` \Rightarrow `cd $TARG`
- ▶ `targ` \Rightarrow `echo $TARG`
- ▶ `settarg -stt` \Rightarrow settarg state stored in environment
- ▶ `settarg -report` \Rightarrow how settarg is configured.

How is settarg connected to Lmod?

- ▶ The settarg command is part of the module command.
- ▶ `module () { eval $($LMOD_CMD bash "$@") && eval $($LMOD_SETTARG_CMD:-:} -s sh) }`
- ▶ Normally `$LMOD_SETTARG_CMD` is “:” or “” so 2nd cmd is a no-op.
- ▶ With settarg loaded it becomes: `$LMOD_DIR/settarg_cmd`

What do dbg/opt/mdbg do?

- ▶ They all set \$TARG_BUILD_SCENARIO
- ▶ It is up to the Makefile to interpret
- ▶ dbg \Rightarrow CFLAGS = -g -O0
- ▶ opt \Rightarrow CFLAGS = -O3
- ▶ mdbg \Rightarrow CFLAGS = -g -O0 and array subscript checking

Show examples

- ▶ `dbg/opt/mdbg` \Rightarrow `$TARG`, `$PATH`
- ▶ `ml -impi` \Rightarrow `$TARG`
- ▶ `ml -intel` \Rightarrow `$TARG`

Example w/o Makefile changes

- ▶ `cd xalt; mkdir -p $TARG; cdt;`
- ▶ `../../configure ...`
- ▶ `make install`

contrib/settarg/make_example/Make-file.simple

```
ifeq ($(TARG_COMPILER_FAMILY),gcc)
    CC := gcc
endif
ifeq ($(TARG_COMPILER_FAMILY),intel)
    CC := icc
endif
ifneq ($(TARG),)
    override O_DIR := $(TARG)/
endif

EXEC := $(O_DIR)hello
SRC  := main.c hello.c
OBJS := $(O_DIR)main.o $(O_DIR)hello.o

all: $(O_DIR) $(EXEC)
$(O_DIR):
    mkdir -p $(O_DIR)
$(EXEC): $(OBJS)
    $(LINK.c) -o $@ $^

$(O_DIR)%.o : %.c
    $(COMPILE.c) -o $@ -c $<

$(O_DIR)main.o : main.c hello.h
$(O_DIR)hello.o: hello.c hello.h
```

Show examples with *Makefile.simple*

- ▶ `dbg; make -f Makefile.simple; type hello`
- ▶ `opt; make -f Makefile.simple; type hello`
- ▶ `ml gcc; dbg; make -f Makefile.simple; type hello`

Show examples with Makefile

- ▶ `rm -rf OBJ/`
- ▶ `dbg; make -f Makefile; type hello`
- ▶ `opt; make -f Makefile; type hello`
- ▶ `ml gcc; dbg; make -f Makefile.simple; type hello`

Personal/Directory setting of settarg

- ▶ Loading order of all settarg config files:
- ▶ System settarg lmod/settarg/settarg_rc.lua
- ▶ ~/.settarg.lua
- ▶ Current or parent directory .settarg.lua
- ▶ all are loaded with over-write of table entries.

settarg -report

- ▶ Reports the current state of rules for settarg
- ▶ Show example with a directory .settarg.lua

a directory .settarg.lua

```
TargetList = { "mach", "build_scenario", "compiler",  
               "mpi", "solver", "file_io "}
```

Show example of \$TARG with above .settarg.lua

- ▶ \$TARG
- ▶ cd w/dao; \Rightarrow \$TARG
- ▶ ml petsc; \Rightarrow \$TARG

Conclusions for Settarg modules

- ▶ A way to switch between different kinds of builds:
dbg/opt/mdbg
- ▶ Avoiding make clean in-between.
- ▶ Highly customizable for your needs.
- ▶ Can be made to work where you have settarg and other don't
- ▶ More detail:
https://lmod.readthedocs.io/en/latest/310_settarg.html

Future Topics

- ▶ Lmod Testing System?
- ▶ More internals of Lmod?
- ▶ collections?
- ▶ Guest Presentation of special issues?