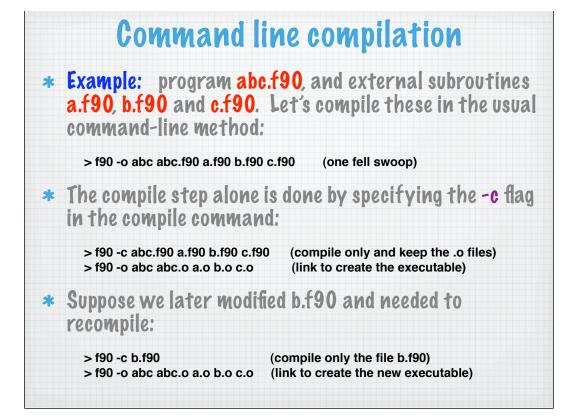
Building Your Code and an Introduction to Makefiles

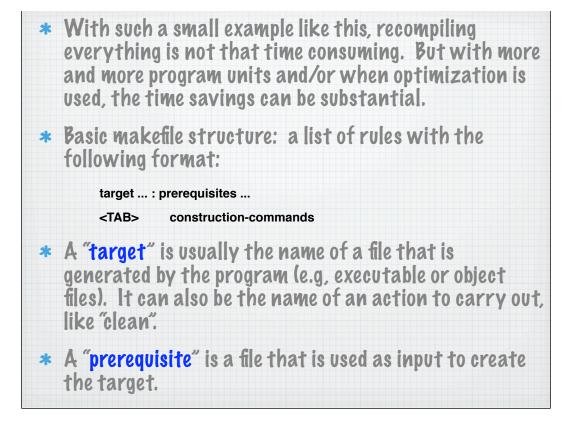


Introduction to Makefiles

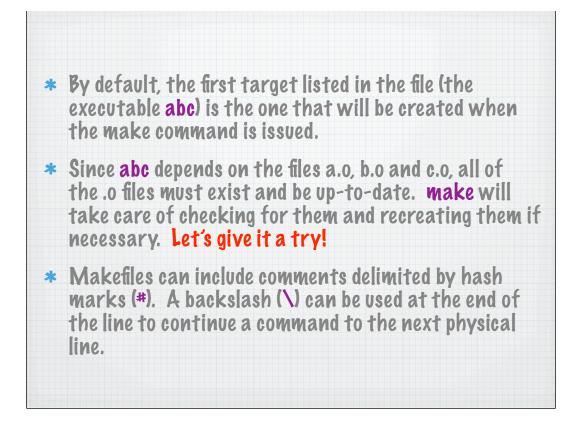
* A makefile is just a set of rules to determine which pieces of a large program need to be recompiled, and issues commands to recompile them.

* For large programs, it's usually convenient to keep each program unit in a separate file. Keeping all program units in a single file is impractical because a change to a single subroutine requires recompilation of the entire program, which can be time consuming.

* When changes are made to some of the source files, only the updated files need to be recompiled, although all relevant files must be linked to create the new executable.



# makefile : makes the	ABC program		
abc : a.o b.o c.o abc.o			
f90 -o abc abc.o a.o	0.0 C.O		
abc.o : abc.f90			
f90 -O -c abc.f90			
a.o : a.f90			
f90 -O -c a.f90			
b.o : b.f90			
f90 -O -c b.f90			
c.o : c.f90			
f90 -O -c c.f90			



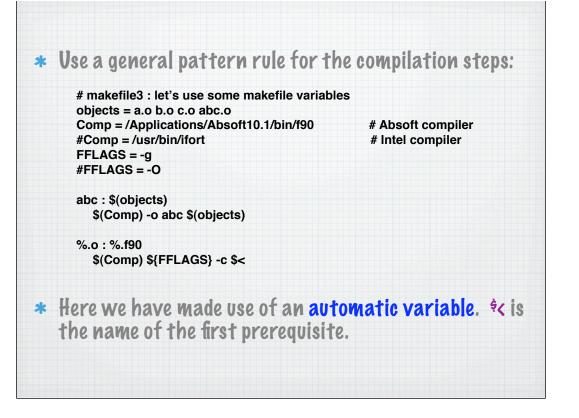
How Poes Make Work?

- * The make utility compares the modification time of the target file with the modification times of the prerequisite files. Any prerequisite file that has a more recent modification time than its target file forces the target file to be recreated.
- * By default, the first target file appearing in the makefile is the one that is built. Other targets are checked only if they are prerequisites for the initial target.
- Other than the fact that the first target in the makefile is the default, the order of the targets does not matter. The make utility will build them in the order required.

More MAKE Functionality

- * By default, if you just type make, then the make utility looks for a file called makefile or Makefile. Use make -f <mymakefile> to explicitly define what file to use.
- Semake -n to display which commands will be used to build the program but it will not actually execute them.
- * Use a "phony target" to code in a clean-up section.
- * Define variables for compiler type, compiler flags, list of all objects, etc. Note that variables names in make are case-sensitive!

* example: makefile2



More complexity

- * Handling c-preprocessor statements.
 - most compilers automatically process .F90 and .F files
- * External libraries like netcdf.
- * Separate compile options for certain files.
- * Placing object files into archives.



- 1. The order of compilation matters! If module B uses module A, then module A must be compiled first. This means that developing and maintaining dependency lists can be a bit cumbersome.
- * There are some free scripts that try to handle this for you. EXAMPLE: fmkmf.pl
- 2. Most Fortran compilers generate module files in addition to object files.
- 3. Make is general-purpose: works great with C, but not so great with complex Fortran codes.

Foray: an alternative to make

- * Designed specifically to handle Fortran builds.
- * Simple to install, uses just one file (buildinfo).
- * Supports multiple interdependent targets
- * Supports mutiple build configurations (debug, release, parallel, serial).
- * Understands fortran dependencies and determines them automatically.
- * Ability to set different compile options for different groups of files or individual files.
- Uses archives in place of object files, and stores time stamps in a separate database.