Follow along with the examples…

```
[(it_nss:frey)@login00.darwin ~]$ mkdir ~/sw

[(it_nss:frey)@login00.darwin ~]$ cd ~/sw

[(it_nss:frey)@login00.darwin sw]$ git clone https://gitlab.com/jtfrey/unix-software-dev.git

[(it_nss:frey)@login00.darwin sw]$ cd unix-software-dev

[(it_nss:frey)@login00.darwin unix-software-dev]$ ls -l
total 124
-rw-r--r-- 1 frey everyone 982 Sep 25 17:32 README.md
drwxr-xr-x 2 frey everyone  15 Sep 25 17:32 src-1
drwxr-xr-x 4 frey everyone   8 Sep 25 17:32 src-2
drwxr-xr-x 2 frey everyone   8 Sep 25 17:32 src-3
drwxr-xr-x 2 frey everyone   7 Sep 25 17:32 src-4
drwxr-xr-x 4 frey everyone   6 Sep 25 17:32 src-5
drwxr-xr-x 4 frey everyone   5 Sep 25 17:32 valet-2.1
```

UD IT Research Cyberinfrastructure

HPC Software Development & Installation

HPC Software Development & Installation

Topics that will be covered:

- Software project kind & scope
- General organizational principles
- Compiled software projects
  - Simple Makefile infrastructure
  - GNU autoconf
  - CMake

Additional topics, time permitting:

- Python virtual environments

Software Project Kind & Scope

- Varying degrees of scale to programming projects
  - Tool programs
    - From a shell or Perl script to extract key data from an output file…
    - …to a multi-file Fortran or Python program that post-processes (via computation) data from an output file

Software Project Kind & Scope

- Varying degrees of scale to programming projects
  - Tool programs
  - Code libraries
    - From a simple Unix archive file (e.g. libcompute.a) containing compiled object code…
    - …to a dynamic shared library (e.g. libcompute.so) with a strong API exposed via header files.

- Many scripting (non-compiled) languages also have the concept of code libraries
  - Matlab .m files that add functions to the environment
  - Python modules (e.g. see /usr/lib64/python2.6/site-packages on Farber)
  - Perl modules (e.g. see /usr/lib64/perl5 on Farber)
- Note that creating an API demands more planning and structure to a project

Software Project Kind & Scope

- Varying degrees of scale to programming projects
  - Tool programs
  - Code libraries
    - From a sim[…] compiled object code…
    - …to a dyna[…] API exposed via header file[…]

An application program interface (API) is a set of routines, protocols, and tools for building software applications.

An API specifies how software components should interact.

Software Project Kind & Scope

- Varying degrees of scale to programming projects
  - Tool programs
  - Code libraries
  - Software suites
    - Containing a mix of tools and libraries

Software Project Kind & Scope

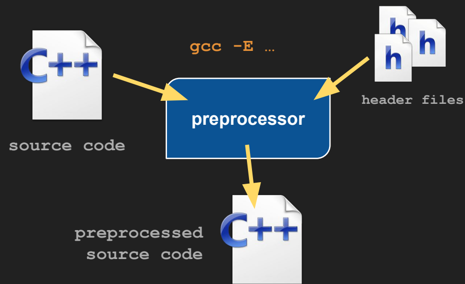
- Varying degrees of scale to programming projects
- A project that starts at the simpler end of the scale can evolve toward the complex end…
- …or it could move between the types
  - a collection of tool programs ⇒ library
  - a very complex tool program ⇒ software suite

The Software Build Process

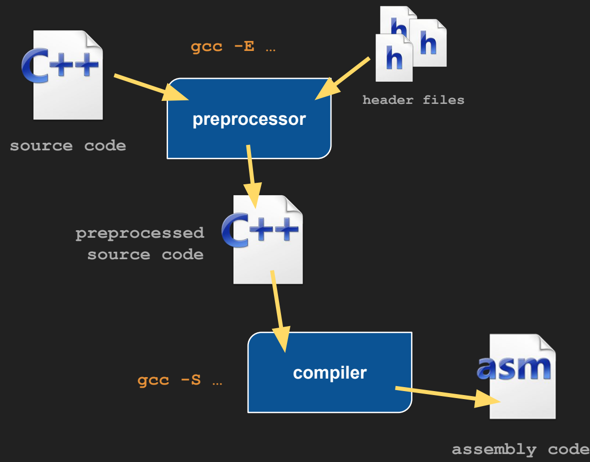
The Software Build Process

source code

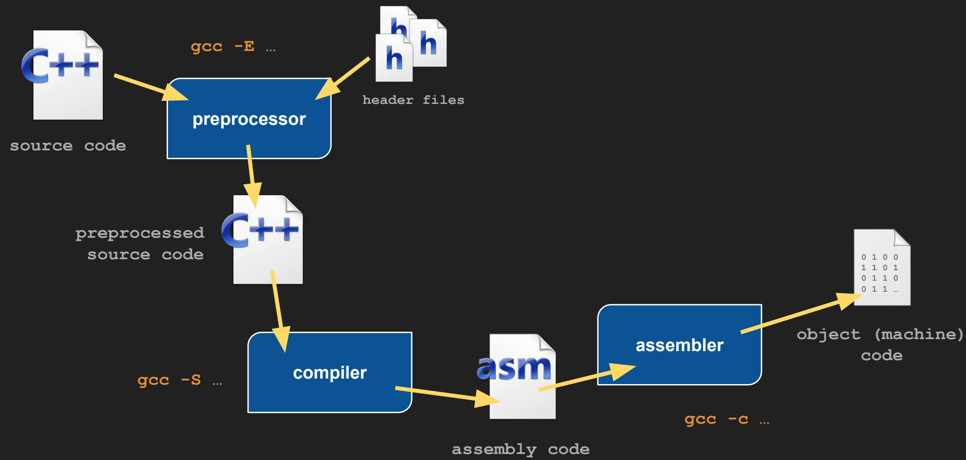
The Software Build Process

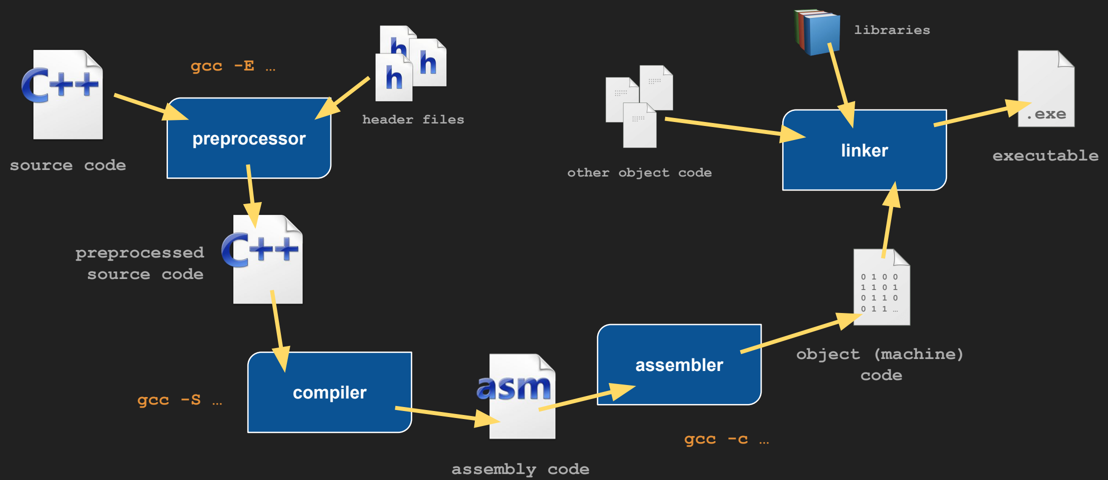


The Software Build Process



The Software Build Process




The Software Build Process
source code
gcc -E …
preprocessor
header files
preprocessed
source code
compiler
gcc -S …
assembly code
assembler
gcc -c …
object (machine)
code
other object code
libraries
linker
executable
.exe
UD IT Research Cyberinfrastructure

The Software Build Process: An Exercise

- Note the `gcc` commands cited on the previous slide
  - Output the preprocessed source and go no further…
  - Output the assembly and go no further…

```
[(it_nss:frey)@login00.darwin ~]$ cat <<EOT > exercise.c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    int       i = rand();

    i = i + 10;
    printf("i = %d\n", i);
    return 0;
}

EOT

[(it_nss:frey)@login00.darwin ~]$ gcc -E -o exercise.i exercise.c
[(it_nss:frey)@login00.darwin ~]$ gcc -S -o exercise.s exercise.i
[(it_nss:frey)@login00.darwin ~]$ gcc -c -o exercise.o exercise.s
[(it_nss:frey)@login00.darwin ~]$ gcc    -o exercise   exercise.o

[(it_nss:frey)@login00.darwin ~]$ ./exercise
i = 1804289393
```

The Software Build Process: An Exercise

- Note the `gcc` commands cited on the previous slide
  - Output the preprocessed source and go no further…
  - Output the assembly and go no further…
- Examine the preprocessed (.i) and assembly (.s) files
  - Still textual program code
- Examine the object code with `objdump -d exercise.o`
- Examine the executable with `readelf -a exercise`
- Explore the other options available with these utilities (man pages)

```
[(it_nss:frey)@login00.darwin ~]$ gcc -E -o exercise.i exercise.c
[(it_nss:frey)@login00.darwin ~]$ gcc -S -o exercise.s exercise.i
[(it_nss:frey)@login00.darwin ~]$ gcc -c -o exercise.o exercise.s
[(it_nss:frey)@login00.darwin ~]$ gcc    -o exercise   exercise.o
```

The Software Build Process: An Exercise

- No
  -
  -
- Ex
  -
- Ex
- Examine the e... ...ble with `readelf -a exercise`
- Explore the other options available with these utilities (man pages)

Any sufficiently advanced technology is indistinguishable from magic.

Arthur C. Clarke's Third Law

```
[(it_nss:frey)@login00.darwin ~]$ gcc -E -o exercise.i exercise.c
[(it_nss:frey)@login00.darwin ~]$ gcc -S -o exercise.s exercise.i
[(it_nss:frey)@login00.darwin ~]$ gcc -c -o exercise.o exercise.s
[(it_nss:frey)@login00.darwin ~]$ gcc    -o exercise   exercise.o
```

The Software Build Process: An Exercise

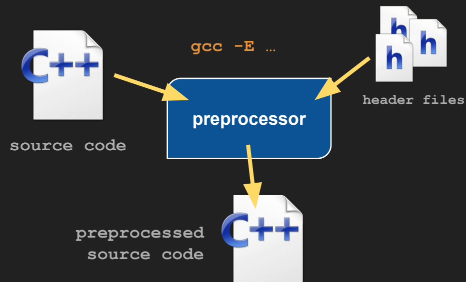
- No
  -
  -
- Ex
  -
- Ex
- Examine the e... ble with `readelf -a exercise`
- Explore the other options available with these utilities (man pages)

For a scientist, any technology that appears magical should prompt curiosity and careful investigation.

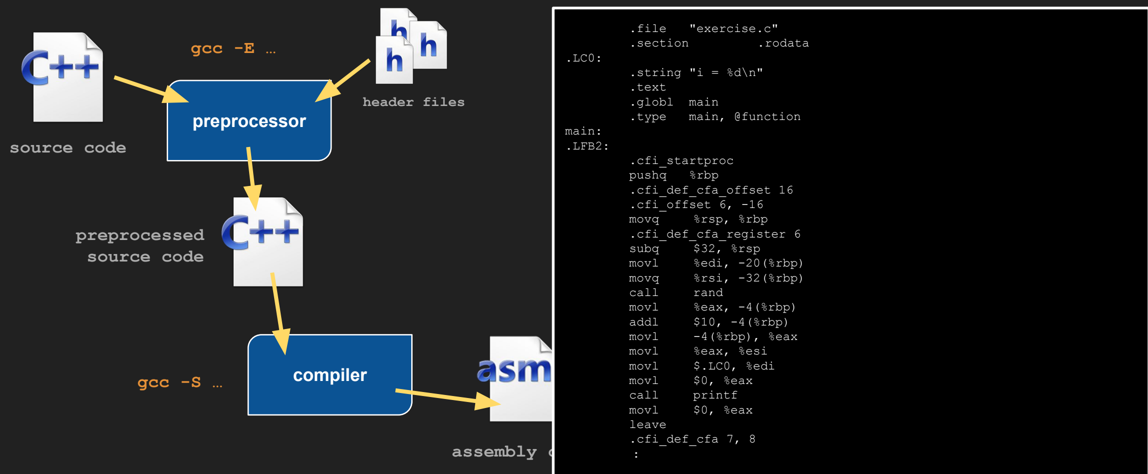
Frey's Corollary to the Third Law

```
[(it_nss:frey)@login00.darwin ~]$ gcc -E -o exercise.i exercise.c
[(it_nss:frey)@login00.darwin ~]$ gcc -S -o exercise.s exercise.i
[(it_nss:frey)@login00.darwin ~]$ gcc -c -o exercise.o exercise.s
[(it_nss:frey)@login00.darwin ~]$ gcc    -o exercise   exercise.o
```

The Software Build Process



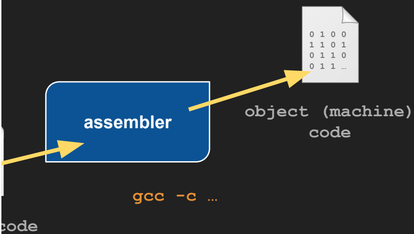

```
# 1 "exercise.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "exercise.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 375 "/usr/include/features.h" 3 4
# 1 "/usr/include/sys/cdefs.h" 1 3 4
# 392 "/usr/include/sys/cdefs.h" 3 4
# 1 "/usr/include/bits/wordsize.h" 1 3 4

   :

# 952 "/usr/include/stdlib.h" 2 3 4
# 964 "/usr/include/stdlib.h" 3 4

# 3 "exercise.c" 2

int main(int argc, char* argv[])
{
        int i = rand();

        i = i + 10;
        printf("i = %d\n", i);
        return 0;
}
```



- Note that the code is now specific to the CPU on which you are building the software — the target ISA

The Software Build Process

```
[frey@login01.darwin test]$ objdump -d exercise.o

exercise.o:      file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0:    55                      push   %rbp
   1:    48 89 e5                mov      %rsp,%rbp
   4:    48 83 ec 20             sub      $0x20,%rsp
   8:    89 7d ec                mov      %edi,-0x14(%rbp)
   b:    48 89 75 e0             mov      %rsi,-0x20(%rbp)
   f:    e8 00 00 00 00          callq  14 <main+0x14>
  14:    89 45 fc                mov      %eax,-0x4(%rbp)
  17:    83 45 fc 0a             addl   $0xa,-0x4(%rbp)
  1b:    8b 45 fc                mov      -0x4(%rbp),%eax
  1e:    89 c6                   mov      %eax,%esi
  20:    bf 00 00 00 00          mov      $0x0,%edi
  25:    b8 00 00 00 00          mov      $0x0,%eax
  2a:    e8 00 00 00 00          callq  2f <main+0x2f>
  2f:    b8 00 00 00 00          mov      $0x0,%eax
  34:    c9                      leaveq
  35:    c3                      retq
```



code

gcc -c …

General Organizational Principles

General Organizational Principles

- Any OS to which you've been exposed probably has the same solution to organizing files: a directory tree
  - Top-most directory is most generic, sub-directory levels become increasingly specific

General Organizational Principles: Installed Software

- Top directory contains all *versions* or *variants* of the installed software
  - version: a frozen, point-in-time snapshot of software, often with a monotonically-increasing tiered numbering scheme (e.g. `4.5.1` or `2024.01`)
  - variant: a copy of a software version produced using alternative parameterization or build properties (e.g. `4.5.1` built with Intel versus GCC compilers)

General Organizational Principles: Installed Software

- Top directory contains all *versions* or *variants* of the installed software
- Each version or variant is a directory ideally structured similarly to the Linux file system
  - `bin` directory to hold executable programs
  - `lib` (or `lib64`) directory to hold archive or shared libraries, Python site packages, etc.
  - `include` directory to hold header files (e.g. for an API)
  - `src` if there is source code accompanying the version/variant
    - For software with multiple sources (e.g. library dependencies) unpack each source package under this directory
    - Notable exception is virtualized build systems (some autoconf, CMake)
      - Multiple versions/variants can be built in a single source directory outside the version/variant directory itself

General Organizational Principles: Source Code

- Code is comprised of one or more files (be they source code or interpreted script, etc.)
- Top directory is the project container
  - Typical software project contains source code, documentation, helper scripts, configuration samples, examples
    - Create a directory for each that is required

General Organizational Principles: Source Code

- Code is comprised of one or more files (be they source code or interpreted script, etc.)
- Top directory is the project container

```
[(it_nss:frey)@login00.darwin ~]$ mkdir project

[(it_nss:frey)@login00.darwin ~]$ cd project

[(it_nss:frey)@login00.darwin project]$ mkdir src docs examples

[(it_nss:frey)@login00.darwin project]$ ls -l .
total 11
drwxr-xr-x 2 frey everyone 2 Feb 23 12:40 docs
drwxr-xr-x 2 frey everyone 2 Feb 23 12:40 examples
drwxr-xr-x 2 frey everyone 2 Feb 23 12:40 src
```

UD IT Research Cyberinfrastructure

- On our HPC systems we like to use "s-r-c" as the directory containing source code
- Content (or presence) of docs and examples will depend on each individual project

Building Software: the *make* Utility

Building Software: the *make* Utility

- Each source file depends on zero or more header/module files
  - When a dependency changes, the source file itself has effectively changed
    - `my_program` depends directly on `my_program.c` and `printargv.c`…
    - …any change to `printargv.c` implies change to `my_program`

```
[(it_nss:frey)@login00.darwin ~]$ cd ~/sw/unix-software-dev/src-1

[(it_nss:frey)@login00.darwin src-1]$ ls -l
total 77
-rw-r--r-- 1 frey everyone 115 Sep 25 17:32 Makefile.1
-rw-r--r-- 1 frey everyone 234 Sep 25 17:32 Makefile.2
-rw-r--r-- 1 frey everyone 274 Sep 25 17:32 Makefile.3
-rw-r--r-- 1 frey everyone 240 Sep 25 17:32 Makefile.4
-rw-r--r-- 1 frey everyone 280 Sep 25 17:32 Makefile.5
-rw-r--r-- 1 frey everyone 339 Sep 25 17:32 Makefile.6
-rw-r--r-- 1 frey everyone 113 Sep 25 17:32 my_program.c
-rw-r--r-- 1 frey everyone 195 Sep 25 17:32 printargv.c
-rw-r--r-- 1 frey everyone 105 Sep 25 17:32 printargv.h
-rw-r--r-- 1 frey everyone 497 Sep 25 17:32 README.md
```


- C, C++ languages provide API information via *header* files; Fortran 90 and later use *module* files for this purpose

Building Software: the *make* Utility

- Each source file depends on zero or more header/module files
  - When a dependency changes, the source file itself has effectively changed
    - `my_program` depends directly on `my_program.c` and `printargv.c`…
    - …any change to `printargv.c` implies change to `my_program`

```
[(it_nss:frey)@login00.darwin src-1]$ cat Makefile.1
#
# Makefile for "my_program"
#

my_program: my_program.c printargv.c
	gcc -o my_program my_program.c printargv.c
```

The *product* named `my_program` depends on *ingredients* `my_program.c` and `printargv.c`

- A Makefile contains variable definitions and rules that combine product, ingredients, and a recipe to create the product from the ingredients.

Building Software: the *make* Utility

- Each source file depends on zero or more header/module files
  - When a dependency changes, the source file itself has effectively changed
    - `my_program` depends directly on `my_program.c` and `printargv.c`…
    - …any change to `printargv.c` implies change to `my_program`

```
[(it_nss:frey)@login00.darwin src-1]$ cat Makefile.1
#
# Makefile for "my_program"
#

my_program: my_program.c printargv.c
<TAB>	gcc -o my_program my_program.c printargv.c
```

When a dependency changes, the *product* `my_program` is generated with this *recipe*: compile the two source files and link the object code to produce an executable named `my_program`

-

Building Software: the *make* Utility

- Each source file depends on zero or more header/module files
  - When a dependency changes, the source file itself has effectively changed
    - `my_program` depends directly on `my_program.c` and `printargv.c`…
    - …any change to `printargv.c` implies change to `my_program`

```
[(it_nss:frey)@login00.darwin src-1]$ make -f Makefile.1
gcc -o my_program my_program.c printargv.c

[(it_nss:frey)@login00.darwin src-1]$ ls -ltr
total 78
   :
-rw-r--r-- 1 frey everyone  105 Sep 25 17:32 printargv.h
-rwxr-xr-x 1 frey everyone 8432 Sep 30 16:41 my_program

[(it_nss:frey)@login00.darwin src-1]$ make -f Makefile.1
make: `my_program' is up to date.
```

Building Software: the *make* Utility

- Each source file depends on zero or more header/module files
  - When a dependency changes, the source file itself has effectively changed
    - `my_program` depends directly on `my_program.c` and `printargv.c`…
    - …any change to `printargv.c` implies change to `my_program`

```
[(it_nss:frey)@login00.darwin src-1]$ make -f Makefile.1
make: `my_program' is up to date.

[(it_nss:frey)@login00.darwin src-1]$ touch printargv.c

[(it_nss:frey)@login00.darwin src-1]$ make -f Makefile.1
gcc -o my_program my_program.c printargv.c

[(it_nss:frey)@login00.darwin src-1]$ touch printargv.h

[(it_nss:frey)@login00.darwin src-1]$ make -f Makefile.1
make: `my_program' is up to date.
```

The **`touch`** command alters the modification timestamp on a file — now `printargv.c` has "changed"

- Why does "changing" printargv.h not trigger a rebuild of the program?

Building Software: the *make* Utility

- The make utility is generally useful — not just for compiling code

```
[(it_nss:frey)@login00.darwin www]$ cat Makefile

manual.txt: man-sbatch.txt man-sacct.txt
        cat man-sbatch.txt man-sacct.txt > manual.txt

man-sbatch.txt:
        man -E ascii sbatch > man-sbatch.txt

man-sacct.txt:
        man -E ascii sacct > man-sacct.txt


[(it_nss:frey)@login00.darwin www]$ make
```


- What happens when I execute the "make" command in this otherwise-empty directory?

Building Software: the *make* Utility

- The make utility is generally useful — not just for compiling code

```
[(it_nss:frey)@login00.darwin www]$ cat Makefile

manual.txt: man-sbatch.txt man-sacct.txt
        cat man-sbatch.txt man-sacct.txt > manual.txt

man-sbatch.txt:
        man -E ascii sbatch > man-sbatch.txt

man-sacct.txt:
        man -E ascii sacct > man-sacct.txt


[(it_nss:frey)@login00.darwin www]$ make
man -E ascii sbatch > man-sbatch.txt
man -E ascii sacct > man-sacct.txt
cat man-sbatch.txt man-sacct.txt > manual.txt


[(it_nss:frey)@login00.darwin www]$ cat manual.txt
sbatch(1)                                  Slurm Commands                                  sbatch(1)



NAME
        sbatch - Submit a batch script to Slurm.
           :
```

First *rule* is the *default rule* for the `Makefile`

With no `-f <filename>` the default `Makefile`/`makefile` is tried

-

Building Software: the *make* Utility

- The make utility is generally useful — not just for compiling code

```
[(it_nss:frey)@login00.darwin www]$ rm manual.txt

[(it_nss:frey)@login00.darwin www]$ make
```


- Removing the manual.txt file — the default product — what will happen when I execute "make?"

Building Software: the *make* Utility

- The make utility is generally useful — not just for compiling code

```
[(it_nss:frey)@login00.darwin www]$ rm manual.txt


[(it_nss:frey)@login00.darwin www]$ make
cat man-sbatch.txt man-sacct.txt > manual.txt
```


- No need to regenerate the two component .txt files, so the two files are concatenated to produce manual.txt

Building Software: the *make* Utility

- The make utility is generally useful — not just for compiling code
  - Variables, automatic variables, and patterned rules simplify a Makefile

```
[(it_nss:frey)@login00.darwin www]$ cat Makefile

MAN_CMD     = man -E ascii
COMPONENTS  = man-sbatch.txt man-sacct.txt

manual.txt: $(COMPONENTS)
        cat $+ > $@

man-%.txt:
        $(MAN_CMD) $* > $@


[(it_nss:frey)@login00.darwin www]$ make
man -E ascii sbatch > man-sbatch.txt
man -E ascii sacct > man-sacct.txt
cat man-sbatch.txt man-sacct.txt > manual.txt
```


-

Building Software: the *make* Utility

- The make utility is generally useful — not just for compiling code
  - Variables, automatic variables, and patterned rules simplify a Makefile

```
[(it_nss:frey)@login00.darwin www]$ cat Makefile

MAN_CMD     = man -E ascii
COMPONENTS  = man-sbatch.txt man-sacct.txt

manual.txt: $(COMPONENTS)
	cat $+ > $@

man-%.txt:
	$(MAN_CMD) $* > $@


[(it_nss:frey)@login00.darwin www]$ make
man -E ascii sbatch > man-sbatch.txt
man -E ascii sacct > man-sacct.txt
cat man-sbatch.txt man-sacct.txt > manual.txt
```

Use variables for repeated expressions, lists

Applies to products whose name match the pattern (something substituted for `%`)

-

Building Software: the *make* Utility

- The make utility is generally useful — not just for compiling code
  - Variables, automatic variables, and patterned rules simplify a Makefile

```
[(it_nss:frey)@login00.darwin www]$ cat Makefile

   CMD      = man -E ascii
   ONENTS   = man-sbatch.txt man-sacct.txt

   l.txt: $(COMPONENTS)
          $+ > $@

man-%.txt:
        $(MAN_CMD) $* > $@



[(it_nss:frey)@login00.darwin www]$ make
man -E ascii sbatch > man-sbatch.txt
man -E ascii sacct > man-sacct.txt
cat man-sbatch.txt man-sacct.txt > manual.txt
```

Automatic variables = captured parts of the rule

-

Building Software: the *make* Utility

- Improving rebuild workload
  - Retain compiler output — *object code* — as intermediates used in linking

```
[(it_nss:frey)@login00.darwin src-1]$ cat Makefile.2
#
# Makefile for "my_program"
#

my_program: my_program.o printargv.o
        gcc -o my_program my_program.o printargv.o

my_program.o: my_program.c printargv.h
        gcc -c my_program.c

printargv.o: printargv.c printargv.h
        gcc -c printargv.c
```

The *product* named `my_program` depends on *ingredients* `my_program.o` and `printargv.o`

The *product* named `my_program.o` depends on *ingredients* `my_program.c` and `printargv.h`

-

Building Software: the *make* Utility

- Improving rebuild workload
  - Retain compiler output — *object code* — as intermediates used in linking

```
[(it_nss:frey)@login00.darwin src-1]$ cat Makefile.2
#
# Makefile for "my_program"
#

my_program: my_program.o printargv.o
        gcc -o my_program my_program.o printargv.o

my_program.o: my_program.c printargv.h
        gcc -c my_program.c

printargv.o: printargv.c printargv.h
        gcc -c printargv.c
```

The *product* named `my_program` depends on *ingredients* `my_program.o` and `printargv.o`

The *product* named `my_program.o` depends on *ingredients* `my_program.c` and `printargv.h`

CHANGED `my_program.c` → CHANGED `my_program.o` → CHANGED `my_program`

UD IT Research Cyberinfras

- What gets rebuilt when printargv.h has changed?

Building Software: the *make* Utility

- Improving rebuild workload
  - Retain compiler output — *object code* — as intermediates used in linking

```
[frey@login00.darwin src-1]$ make -f Makefile.2
gcc -c my_program.c
gcc -c printargv.c
gcc -o my_program my_program.o printargv.o

[frey@login00.darwin src-1]$ touch printargv.h
[frey@login00.darwin src-1]$ make -f Makefile.2
gcc -c my_program.c
gcc -c printargv.c
gcc -o my_program my_program.o printargv.o

[frey@login00.darwin src-1]$ touch printargv.c
[frey@login00.darwin src-1]$ make -f Makefile.2
gcc -c printargv.c
gcc -o my_program my_program.o printargv.o

[frey@login00.darwin src-1]$ ls -ltr
    :
-rw-r--r-- 1 frey everyone 1392 Oct 11 11:29 my_program.o
-rw-r--r-- 1 frey everyone  195 Oct 11 11:29 printargv.c
-rw-r--r-- 1 frey everyone 1560 Oct 11 11:29 printargv.o
-rwxr-xr-x 1 frey everyone 8432 Oct 11 11:29 my_program
```


-

Building Software: the *make* Utility

- Decrease repetition, increase readability
  - Use *variables* for long or repeated values in the Makefile text

```
[frey@login00.darwin src-1]$ cat Makefile.3
#
# Makefile for "my_program"
#

TARGET          = my_program

OBJECTS         = printargv.o my_program.o

$(TARGET): $(OBJECTS)
        gcc -o $(TARGET) $(OBJECTS)

my_program.o: my_program.c printargv.h
        gcc -c my_program.c

printargv.o: printargv.c printargv.h
        gcc -c printargv.c
```


•

Building Software: the *make* Utility

- Decrease repetition, increase readability
  - Use *variables* for long or repeated values in the Makefile text

```
[frey@login00.darwin src-1]$ cat Makefile.3
#
# Makefile for "my_program"
#

TARGET          = my_program

OBJECTS         = printargv.o my_progra

$(TARGET): $(OBJECTS)
        gcc -o $(TARGET) $(OBJECTS)

my_program.o: my_program.c printargv.h
        gcc -c my_program.c

printargv.o: printargv.c printargv.h
        gcc -c printargv.c
```

Variable value is referenced using name inside parentheses — $TARGET = $(T)ARGET

-

Building Software: the *make* Utility

- Decrease repetition, increase readability
  - *Automatic variables* yield components of the matched rule

```
[frey@login00.darwin src-1]$ cat Makefile.4
#
# Makefile for "my_program"
#

TARGET          = my_program

OBJECTS         = printargv.o my_program.o

$(TARGET): $(OBJECTS)
	gcc -o $@ $+

my_program.o: my_program.c printargv.h
	gcc -c $<

printargv.o: printargv.c printargv.h
	gcc -c $<
```


-

Building Software: the *make* Utility

- Decrease repetition, increase readability
  - *Automatic variables* yield components of the matched rule

```
[frey@login00.darwin src-1]$ cat Makefile.4
#
# Makefile for "my_program"
#

TARGET          = my_program

OBJECTS         = printa

$(TARGET): $(OBJECTS)
        gcc -o $@ $+

my_program.o: my_prog
        gcc -c $<

printargv.o: printarg
        gcc -c $<
```

$@ = the *product* in the rule
$+ = all *ingredients* in the rule

$< = the first *ingredient* in the rule's list of ingredients

- Google for the term "makefile automatic variables" for more information on what's available

Building Software: the *make* Utility

- Pattern-based rules
  - Another means of avoiding repetition, increasing readability

```
[frey@login00.darwin src-1]$ cat Makefile.5
#
# Makefile for "my_program"
#

TARGET          = my_program

OBJECTS         = printargv.o my_program.o

$(TARGET): $(OBJECTS)
        gcc -o $@ $+ $(LDFLAGS) $(LIBS)

my_program.o: my_program.c printargv.h

printargv.o: printargv.c printargv.h

%.o: %.c
        gcc -c $(CPPFLAGS) $(CFLAGS) $<
```


- Automatic variables are absolutely necessary for pattern-based rules

Building Software: the *make* Utility

- Pattern-based rules
  - Another means of avoiding repetition, increasing readability

```
[frey@login00.darwin src-1]$ cat Makefile.5
#
# Makefile for "my_program"
#

TARGET          = my_program

OBJECTS         = printargv.o my_program.o

$(TARGET): $(OBJECTS)
        gcc -o $@ $+ $(LDFLAGS) $(LIBS)

my_program.o: my_program.c printargv.h

printargv.o: printargv.c printargv.h

%.o: %.c
        gcc -c $(CPPFLAGS) $(CFLAGS) $<
```

Dependencies cited alone…

…matched with a pattern-based rule to build product from ingredient(s)

-

Building Software: the *make* Utility

- Additional targets
  - Rules that effect other changes to the build products et al.

```
[frey@login00.darwin src-1]$ cat Makefile.6
#
# Makefile for "my_program"
#

TARGET          = my_program

OBJECTS         = printargv.o my_program.o

default: $(TARGET)

clean:
	$(RM) $(TARGET) $(OBJECTS)

#

$(TARGET): $(OBJECTS)
	gcc -o $@ $+ $(LDFLAGS) $(LIBS)

my_program.o: my_program.c printargv.h

printargv.o: printargv.c printargv.h

%.o: %.c
	gcc -c $(CPPFLAGS) $(CFLAGS) $<
```


-

Building Software: the *make* Utility

- Additional targets
  - Rules that effect other changes to the build products et al.

```
[frey@login00.darwin src-1]$ cat Makefile.6
#
# Makefile for "my_program"
#

TARGET          = my_program

OBJECTS         = pri

default: $(TARGET)

clean:
        $(RM) $(TARGET) $(OBJECTS)

#

$(TARGET): $(OBJECTS)
        gcc -o $@ $+ $(LDFLAGS) $(LIBS)

my_program.o: my_program.c printargv.h

printargv.o: printargv.c printargv.h

%.o: %.c
        gcc -c $(CPPFLAGS) $(CFLAGS) $<
```

First rule = default rule — literally, in this case

The "clean" product has no dependencies, so it always executes: "make clean" removes product(s) and intermediates (object files)

-

Building Software: the *make* Utility

- Additional targets
  - Rules that effect other changes to the build products et al.

```
[frey@login00.darwin src-1]$ make -f Makefile.6 clean
rm -f my_program printargv.o my_program.o


[frey@login00.darwin src-1]$ make -f Makefile.6
gcc -c   printargv.c
gcc -c   my_program.c
gcc -o my_program printargv.o my_program.o


[frey@login00.darwin src-1]$ ./my_program a b c
1        a
2        b
3        c


[frey@login00.darwin src-1]$ cd ../src-2
```


- After all of that, what **does** this program actually do????
- We're now done with the basics of Makefiles

Building Software: the *make* Utility

- Multi-tier source projects — a library and an executable
  - Encapsulate global definitions in separate files

```
[frey@login00.darwin src-2]$ cat Makefile.inc
#
# Makefile.inc
# Global variables for subprojects
#

MAKEFILE_INC    :=$(abspath $(lastword $(MAKEFILE_LIST)))
SRCDIR          :=$(dir $(MAKEFILE_INC))

CC                = gcc
CPPFLAGS        += -DVERSION=1.0
CFLAGS          += -g -O3

LDFLAGS         += -L/usr/lib64
LIBS            += -lm


[frey@login00.darwin src-2]$ cat Makefile.rules
#
# Makefile.rules
# Templated rules used by subprojects
#

%.o: %.c
        gcc -c $(CPPFLAGS) $(CFLAGS) $<
```


-

Building Software: the *make* Utility

- Multi-tier source projects — a library and an executable
  - Encapsulate global definitions in separate files

```
[frey@login00.darwin src-2]$ cat Makefile
#
# Top-level Makefile for subproject
#

SUBPROJS = libprintargv my_program

default:
	@for SUBPROJ in $(SUBPROJS); do make -C $$SUBPROJ; done

clean:
	@for SUBPROJ in $(SUBPROJS); do make -C $$SUBPROJ clean; done
```

For each *word* in SUBPROJS, run the `make` command in a subdirectory of that name — no target cited = default

-

Building Software: the *make* Utility

- Multi-tier source projects — a library and an executable
  - The `include` command inserts another file's content at that location

```
[frey@login00.darwin src-2]$ cat libprintargv/Makefile
#
# Makefile for 'libprintargv' subproject
#

include ../Makefile.inc

TARGET          = libprintargv.a

OBJECTS         = printargv.o

default: $(TARGET)

clean:
	$(RM) $(TARGET) $(OBJECTS)

#

$(TARGET): $(OBJECTS)
	$(AR) cr $(TARGET) $(OBJECTS)

printargv.o: printargv.c printargv.h

include ../Makefile.rules
```

Building Software: the *make* Utility

- Multi-tier source projects — a library and an executable
  - The `include` command inserts another file's content at that location

```
[frey@login00.darwin src-2]$ cat libprintargv/Makefile
#
# Makefile for 'libprintargv' subproject
#

include ../Makefile.inc

TARGET          = libprintargv.a

OBJECTS         = printargv.o

default: $(TARGET)

clean:
	$(RM) $(TARGET) $(OBJECTS)

#

$(TARGET): $(OBJECTS)
	$(AR) cr $(TARGET) $(OBJECTS)

printargv.o: printargv.c printargv.h

include ../Makefile.rules
```

The `ar` utility creates/updates a static archive, a single file containing (in this case) the *object* intermediates

- A *static library* is just a special file containing a collection of object code files emitted by a compiler
- The code in the static library is added directly to any executable that links against it — in contrast to *shared libraries* which only add a reference to the executable
  - When using shared libraries, the library must be present BOTH at build and run time; a static library is not needed at run time

Building Software: the *make* Utility

- Multi-tier source projects — a library and an executable
  - The `include` command inserts another file's content at that location

```
[frey@login00.darwin src-2]$ make
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
gcc -c -DVERSION=1.0 -g -O3 printargv.c
ar cr libprintargv.a printargv.o
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/my_program'
gcc -c -DVERSION=1.0 -I/home/1001/sw/unix-software-dev/src-2//libprintargv -g -O3 my_program.c
gcc -g -O3 -o my_program my_program.o -L/usr/lib64 -L/home/1001/sw/unix-software-dev/src-2//libprintargv -lm -lprintargv
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/my_program'


[frey@login00.darwin src-2]$ ./my_program/my_program a b c
1        a
2        b
3        c
```


- A *static library* is just a special file containing a collection of object code files emitted by a compiler
- The code in the static library is added directly to any executable that links against it — in contrast to *shared libraries* which only add a reference to the executable
  - When using shared libraries, the library must be present BOTH at build and run time; a static library is not needed at run time

Building Software: the *make* Utility

- Exercise: let's add an install target to the build infrastructure
    1. Variables: use a variable named `PREFIX` for the base install path
        - Default to `/usr/local`
    2. Add an "install" target to Makefile that invokes `make install` in subdirectories
    3. Add an "install" target to subdirectories' Makefile
        - libprintargv
            - Copy header file to `$(PREFIX)/include`
            - Copy static library to `$(PREFIX)/lib`
        - my_program
            - Copy executable to `$(PREFIX)/bin`

Building Software: the *make* Utility

1. Variables: use a variable named `PREFIX` for the base install path

```
[frey@login00.darwin src-2]$ vi Makefile.inc
#
# Makefile.inc
# Global variables for subprojects
#

PREFIX          ?= /usr/local

MAKEFILE_INC    :=$(abspath $(lastword $(MAKEFILE_LIST)))
SRCDIR          :=$(dir $(MAKEFILE_INC))

CC              = gcc
CPPFLAGS        += -DVERSION=1.0
CFLAGS          += -g -O3

LDFLAGS         += -L/usr/lib64
LIBS            += -lm
```


●

Building Software: the *make* Utility

1. Variables: use a variable named `PREFIX` for the base install path
2. Add an "install" target to Makefile that invokes `make install` in subdirectories

```
[frey@login00.darwin src-2]$ vi Makefile
#
# Top-level Makefile for subproject
#

SUBPROJS        = libprintargv my_program

default:
	@for SUBPROJ in $(SUBPROJS); do make -C $$SUBPROJ; done

clean:
	@for SUBPROJ in $(SUBPROJS); do make -C $$SUBPROJ clean; done

install:
	@for SUBPROJ in $(SUBPROJS); do make -C $$SUBPROJ install; done
```


•

Building Software: the *make* Utility

1. Variables: use a variable named `PREFIX` for the base install path
2. Add an "install" target to Makefile that invokes `make install` in subdirectories
3. Add an "install" target to subdirectories' Makefile
   - libprintargv

```
[frey@login00.darwin src-2]$ vi libprintargv/Makefile
#
# Makefile for 'libprintargv' subproject
#

include ../Makefile.inc

TARGET    = libprintargv.a

OBJECTS   = printargv.o

default: $(TARGET)

clean:
        $(RM) $(TARGET) $(OBJECTS)

install: $(TARGET)
        cp printargv.h $(PREFIX)/include
        cp $(TARGET) $(PREFIX)/lib

#

$(TARGET): $(OBJECTS)
        $(AR) cr $(TARGET) $(OBJECTS)

printargv.o: printargv.c printargv.h

include ../Makefile.rules
```


-

Building Software: the *make* Utility

1. Variables: use a variable named `PREFIX` for the base install path
2. Add an "install" target to Makefile that invokes `make install` in subdirectories

This will fail because the destination directories do not necessarily exist!

   - libprintargv

```
[frey@login00.darwin src-2]$ vi libprintargv/Makefile
#
# Makefile for 'libprintargv' subproject
#

include ../Makefile.inc

TARGET    = libprintargv.a

OBJECTS   = printargv.o

default: $(TARGET)

clean:
	$(RM) $(TARGET) $(OBJECTS)

install: $(TARGET)
	cp printargv.h $(PREFIX)/include
	cp $(TARGET) $(PREFIX)/lib

#

$(TARGET): $(OBJECTS)
	$(AR) cr $(TARGET) $(OBJECTS)

printargv.o: printargv.c printargv.h

include ../Makefile.rules
```


- Could just also include "mkdir" commands in the "install" recipe, but why not leverage make dependencies to ONLY do so when necessary?

Building Software: the *make* Utility

1. Variables: use a variable named `PREFIX` for the base install path
2. Add an "install" target to Makefile that invokes `make install` in subdirectories
3. Add an "install" target to subdirectories' Makefile
   - libprintargv

```
[frey@login00.darwin src-2]$ vi libprintargv/Makefile
#
# Makefile for 'libprintargv' subproject
#

include ../Makefile.inc

TARGET    = libprintargv.a

OBJECTS   = printargv.o

default: $(TARGET)

clean:
        $(RM) $(TARGET) $(OBJECTS)

$(PREFIX)/include:
        mkdir -p $@

$(PREFIX)/lib:
        mkdir -p $@

install: $(TARGET) $(PREFIX)/include $(PREFIX)/lib
        cp printargv.h $(PREFIX)/include
        cp $(TARGET) $(PREFIX)/lib

#

$(TARGET): $(OBJECTS)
        $(AR) cr $(TARGET) $(OBJECTS)
  :
```


•

Building Software: the *make* Utility

1. Variables: use a variable named `PREFIX` for the base install path
2. Add an "install" target to Makefile that invokes `make install` in subdirectories
3. Add an "install" target to subdirectories' Makefile
   - libprintargv
   - my_program

```
[frey@login00.darwin src-2]$ vi my_program/Makefile
#
# Makefile for 'my_program' subproject
#

include ../Makefile.inc

TARGET                   = my_program

OBJECTS                  = my_program.o

CPPFLAGS                 += -I$(SRCDIR)/libprintargv
LDFLAGS                  += -L$(SRCDIR)/libprintargv
LIBS                     += -lprintargv

default: $(TARGET)

clean:
        $(RM) $(TARGET) $(OBJECTS)

$(PREFIX)/bin:
        mkdir -p $@

install: $(TARGET) $(PREFIX)/bin
        cp $(TARGET) $(PREFIX)/bin

#
$(TARGET): $(OBJECTS)
        $(CC) $(CFLAGS) -o $@ $+ $(LDFLAGS) $(LIBS)

   :
```


•

Building Software: the *make* Utility

- Give it a try!

```
[frey@login00.darwin src-2]$ make clean
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
rm -f libprintargv.a printargv.o
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/my_program'
rm -f my_program my_program.o
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/my_program'


[frey@login00.darwin src-2]$ make
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
gcc -c -DVERSION=1.0 -g -O3 printargv.c
ar cr libprintargv.a printargv.o
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/my_program'
gcc -c -DVERSION=1.0 -I/home/1001/sw/unix-software-dev/src-2//libprintargv -g -O3 my_program.c
gcc -g -O3 -o my_program my_program.o -L/usr/lib64 -L/home/1001/sw/unix-software-dev/src-2//libprintargv -lm -lprintargv
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/my_program'
```


-

Building Software: the *make* Utility

- Give it a try!

```
[frey@login00.darwin src-2]$ make install
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
cp printargv.h /usr/local/include
cp: cannot create regular file '/usr/local/include/printargv.h': Permission denied
make[1]: *** [install] Error 1
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/my_program'
cp my_program /usr/local/bin
cp: cannot create regular file '/usr/local/bin/my_program': Permission denied
make[1]: *** [install] Error 1
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/my_program'
make: *** [install] Error 2
```


-

Building Software: the *make* Utility

- Give it a try!

```
[frey@login00.darwin src-2]$ make PREFIX=/tmp/xyz install
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
mkdir -p /tmp/xyz/include
mkdir -p /tmp/xyz/lib
cp printargv.h /tmp/xyz/include
cp libprintargv.a /tmp/xyz/lib
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/my_program'
mkdir -p /tmp/xyz/bin
cp my_program /tmp/xyz/bin
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/my_program'


[frey@login00.darwin src-2]$ /tmp/xyz/bin/my_program a b c
1        a
2        b
3        c


[frey@login00.darwin src-2]$ ls -l /tmp/xyz/include
total 4
-rw-r--r-- 1 frey everyone 105 Oct 14 13:30 printargv.h


[frey@login00.darwin src-2]$ ls -l /tmp/xyz/lib
total 8
-rw-r--r-- 1 frey everyone 6426 Oct 14 13:30 libprintargv.a
```


-

Building Software: the *make* Utility

- What if we "change" a source file?

```
[frey@login00.darwin src-2]$ touch libprintargv/printargv.c


[frey@login00.darwin src-2]$ make PREFIX=/tmp/xyz install
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
gcc -c -DVERSION=1.0 -g -O3 printargv.c
ar cr libprintargv.a printargv.o
cp printargv.h /tmp/xyz/include
cp libprintargv.a /tmp/xyz/lib
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/my_program'
gcc -c -DVERSION=1.0 -I/home/1001/sw/unix-software-dev/src-2//libprintargv -g -O3 my_program.c
gcc -g -O3 -o my_program my_program.o -L/usr/lib64 -L/home/1001/sw/unix-software-dev/src-2//libprintargv -lm -lprintargv
cp my_program /tmp/xyz/bin
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/my_program'
```


- The "install" target doesn't just install, it rebuilds some of the underlying components — what causes "make" to do that??
  - Since the "install" targets depend on the products, when changes affect those products "make" triggers those actions first

Building Software: the *make* Utility

- What if we "change" a source file?

```
[frey@login00.darwin src-2]$ touch libprintargv/printargv.c


[frey@login00.darwin src-2]$ make PREFIX=/tm
make[1]: Entering directory `/home/1001/sw/u
gcc -c -DVERSION=1.0 -g -O3 printargv.c
ar cr libprintargv.a printargv.o
cp printargv.h /tmp/xyz/include
cp libprintargv.a /tmp/xyz/lib
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/my_program'
gcc -c -DVERSION=1.0 -I/home/1001/sw/unix-software-dev/src-2//libprintargv -g -O3 my_program.c
gcc -g -O3 -o my_program my_program.o -L/usr/lib64 -L/home/1001/sw/unix-software-dev/src-2//libprintargv -lm -lprintargv
cp my_program /tmp/xyz/bin
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/my_program'
```

The single source file is recompiled, yielding a new `printargv.o`, which causes the static library to be updated and installed

-

Building Software: the *make* Utility

- What if we "change" a source file?

```
[frey@login00.darwin src-2]$ touch libprintargv/printargv.c


[frey@login00.darwin src-2]$ make PREFIX=/tmp/xyz install
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
gcc -c -DVERSION=1.0 -g -O3 printargv.c
ar cr libprintargv.a printargv.o
cp printargv.h /tmp/xyz/include
cp libprintargv.a /tmp/xyz/lib
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-2/libprintargv'
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-2/my_program'
gcc -c -DVERSION=1.0 -I/home/1001/sw/unix-software-dev/src-2//libprintargv -g -O3 my_program.c
gcc -g -O3 -o my_program my_program.o -L/usr/lib64 -L/home/1001/sw/unix-software-dev/src-2//libprintargv -lm -lprintargv
cp my_program /tmp/xyz/bin
make[
```

The change to the static library causes `my_program` to be relinked — why is this necessary?

-

GNU autotools

GNU autotools

- Writing and maintaining Makefiles by hand can become cumbersome
  - A lot of what we embedded in `Makefile.inc` and `Makefile.rules` is likely common to almost all projects
  - A lot of what we did by hand could be automated
    - For each source file, determine on what other files it depends (e.g. headers)
    - E.g. `gcc … -MMD …` to emit make-style dependency files

GNU autotools

- Writing and maintaining Makefiles by hand can become cumbersome
    - A lot of what we embedded in `Makefile.inc` and `Makefile.rules` is likely common to almost all projects
    - A lot of what we did by hand could be automated
        - For each source file, determine on what other files it depends (e.g. headers)
        - E.g. `gcc … -MMD …` to emit make-style dependency files

```
[frey@login00.darwin src-2]$ make
  :
gcc -c -DVERSION=1.0 -MMD -g -O3 printargv.c
  :
gcc -c -DVERSION=1.0 -MMD -I/home/1001/sw/unix-software-dev/src-2//libprintargv -g -O3 my_program.c
  :


[frey@login00.darwin src-2]$ cat libprintargv/printargv.d
printargv.o: printargv.c printargv.h


[frey@login00.darwin src-2]$ cat my_program/my_program.d
my_program.o: my_program.c \
 /home/1001/sw/unix-software-dev/src-2//libprintargv/printargv.h
```

GNU autotools

- Writing and maintaining Makefiles by hand can become cumbersome
  - A lot of what we embedded in `Makefile.inc` and `Makefile.rules` is likely common to almost all projects
  - A lot of what we did by hand could be automated
    - For each source file, determine on what other files it depends (e.g. headers)
    - E.g. `gcc … -MMD …` to emit make-style dependency files
- Compilers (toolchains) have major differences
  - Detect which toolchain is being used, adapt behavior as a result
    - E.g. automatically generate a `Makefile.inc` to match the *build environment*
  - Detect third-party and OS libraries, functions within them, and adapt accordingly
    - Linux `printf()` behaves differently than Mac OS X `printf()`

UD IT Research Cyberinfrastructure

- The Intel Fortran compiler has different flags versus GCC versus AMD versus

GNU autotools

- Solution: provide a higher-level description of what's needed in the build environment, let the computer solve for the variables
    - GNU autoconf uses configuration files and templates written in the M4 language

```
[frey@login00.darwin src-2]$ cd ../src-3
[frey@login00.darwin src-3]$ cat configure.ac
AC_INIT([Tutorial Program], 1.0)
AM_INIT_AUTOMAKE
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile])
AC_OUTPUT


[frey@login00.darwin src-3]$ autoreconf --install
configure.ac:2: installing './install-sh'
configure.ac:2: installing './missing'
Makefile.am: installing './INSTALL'
Makefile.am: error: required file './NEWS' not found
Makefile.am: error: required file './README' not found
Makefile.am: error: required file './AUTHORS' not found
Makefile.am: error: required file './ChangeLog' not found
Makefile.am: installing './COPYING' using GNU General Public License v3 file
Makefile.am:     Consider adding the COPYING file to the version control system
Makefile.am:     for your code, to avoid questions about which license your project uses
Makefile.am: installing './depcomp'
autoreconf: automake failed with exit status: 1
```


- Name and version of project in AC_INIT
- AC_PROG_CC activates C language handling
- AC_CONFIG_HEADERS writes automake build environment info as macros to to template config.h.in
- AC_CONFIG_FILES turns a Makefile.am template into template Makefile.in

GNU autotools

- Solution: provide a higher-level description of what's needed in the build environment, let the computer solve for the variables
  - GNU autoconf uses configuration files and templates written in the M4 language

```
[frey@login00.darwin src-3]$ touch README NEWS ChangeLog AUTHORS
[frey@login00.darwin src-3]$ autoreconf --install

[frey@login00.darwin src-3]$ ls -ltr
   :
-rw-r--r-- 1 frey everyone  37794 Oct 14 14:29 aclocal.m4
-rw-r--r-- 1 frey everyone625 Oct 14 14:29 config.h.in
-rwxr-xr-x 1 frey everyone  13997 Oct 14 14:29 install-sh
-rwxr-xr-x 1 frey everyone   6873 Oct 14 14:29 missing
-rw-r--r-- 1 frey everyone  15749 Oct 14 14:29 INSTALL
-rw-r--r-- 1 frey everyone  35147 Oct 14 14:29 COPYING
-rwxr-xr-x 1 frey everyone  23566 Oct 14 14:29 depcomp
-rw-r--r-- 1 frey everyone        0 Oct 14 14:30 README
-rw-r--r-- 1 frey everyone        0 Oct 14 14:30 NEWS
-rw-r--r-- 1 frey everyone        0 Oct 14 14:30 ChangeLog
-rw-r--r-- 1 frey everyone        0 Oct 14 14:30 AUTHORS
drwxr-xr-x 2 frey everyone        7 Oct 14 14:30 autom4te.cache
-rwxr-xr-x 1 frey everyone 141680 Oct 14 14:30 configure
-rw-r--r-- 1 frey everyone  23710 Oct 14 14:30 Makefile.in
```


- Generates configure script

GNU autotools

- Solution: provide a higher-level description of what's needed in the build environment, let the computer solve for the variables
  - GNU autoconf uses configuration files and templates written in the M4 language

```
[frey@login00.darwin src-3]$ ./configure --help
`configure' configures Tutorial Program 1.0 to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE.  See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help              display this help and exit
      --help=short        display options specific to this package
      --help=recursive    display the short help of all the included packages
  -V, --version           display version information and exit
  -q, --quiet, --silent   do not print `checking ...' messages
      --cache-file=FILE   cache test results in FILE [disabled]
  -C, --config-cache      alias for `--cache-file=config.cache'
  -n, --no-create         do not create output files
      --srcdir=DIR        find the sources in DIR [configure dir or `..']

Installation directories:
  --prefix=PREFIX         install architecture-independent files in PREFIX
                          [/usr/local]
   :
```


-

GNU autotools

- Solution: provide a higher-level description of what's needed in the build environment, let the computer solve for the variables
    - GNU autoconf uses configuration files and templates written in the M4 language

```
[frey@login00.darwin src-3]$ mkdir build-system ; cd build-system
[frey@login00.darwin build-system]$ CC=gcc ../configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
   :
config.status: creating Makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands


[frey@login00.darwin build-system]$ make
make  all-am
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-3/build-system'
gcc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT printargv.o -MD -MP -MF .deps/printargv.Tpo -c -o printargv.o ../printargv.c
mv -f .deps/printargv.Tpo .deps/printargv.Po
gcc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT my_program.o -MD -MP -MF .deps/my_program.Tpo -c -o my_program.o ../my_program.c
mv -f .deps/my_program.Tpo .deps/my_program.Po
gcc  -g -O2   -o my_program printargv.o my_program.o
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-3/build-system'
```


- Specify the C compiler to use with the "CC" environment variable

GNU autotools

- Solution: provide a higher-level description of what's needed in the build environment, let the computer solve for the variables
  - GNU autoconf uses configuration files and templates written in the M4 language

```
[frey@login00.darwin build-system]$ mkdir ../build-intel ; cd ../build-intel
[frey@login00.darwin build-intel]$ vpkg_require intel/2020
Adding package `intel/2020u4` to your environment

[frey@login00.darwin build-intel]$ CC=icc ../configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
   :
config.status: creating Makefile
config.status: creating config.h
config.status: executing depfiles commands


[frey@login00.darwin build-intel]$ make
make  all-am
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-3/build-intel'
icc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT printargv.o -MD -MP -MF .deps/printargv.Tpo -c -o printargv.o ../printargv.c
mv -f .deps/printargv.Tpo .deps/printargv.Po
icc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT my_program.o -MD -MP -MF .deps/my_program.Tpo -c -o my_program.o ../my_program.c
mv -f .deps/my_program.Tpo .deps/my_program.Po
icc  -g -O2   -o my_program printargv.o my_program.o
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-3/build-intel'
```


- Specify the C compiler to use with the "CC" environment variable

GNU autotools

- Solution: provide a higher-level description of what's needed in the build environment, let the computer solve for the variables
    - GNU autoconf uses configuration files and templates written in the M4 language

```
[frey@login00.darwin build-intel]$ make install
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-3/build-intel'
 /usr/bin/mkdir -p '/usr/local/bin'
  /usr/bin/install -c my_program '/usr/local/bin'
/usr/bin/install: cannot create regular file '/usr/local/bin/my_program': Permission denied
make[1]: *** [install-binPROGRAMS] Error 1
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-3/build-intel'
make: *** [install-am] Error 2
```


- Specify the C compiler to use with the "CC" environment variable

GNU autotools

- Solution: provide a higher-level description of what's needed in the build environment, let the computer solve for the variables
  - GNU autoconf uses configuration files and templates written in the M4 language

```
[frey@login00.darwin build-intel]$ make distclean
[frey@login00.darwin build-intel]$ CC=icc ../configure --prefix=/tmp/abc

[frey@login00.darwin build-intel]$ make install
icc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT printargv.o -MD -MP -MF .deps/printargv.Tpo -c -o printargv.o ../printargv.c
mv -f .deps/printargv.Tpo .deps/printargv.Po
icc -DHAVE_CONFIG_H -I. -I..      -g -O2 -MT my_program.o -MD -MP -MF .deps/my_program.Tpo -c -o my_program.o ../my_program.c
mv -f .deps/my_program.Tpo .deps/my_program.Po
icc  -g -O2   -o my_program printargv.o my_program.o
make[1]: Entering directory `/home/1001/sw/unix-software-dev/src-3/build-intel'
 /usr/bin/mkdir -p '/tmp/abc/bin'
  /usr/bin/install -c my_program '/tmp/abc/bin'
make[1]: Nothing to be done for `install-data-am'.
make[1]: Leaving directory `/home/1001/sw/unix-software-dev/src-3/build-intel'
```


- The "distclean" target removes the build environment that configure generated
- The —prefix option behaves as our PREFIX variable did back in the previous directory's Makefile infrastructure

GNU autotools

- Solution: provide a higher-level description of what's needed in the build environment, let the computer solve for the variables
  - GNU autoconf uses configuration files and templates written in the M4 language

```
[frey@login00.darwin build-intel]$ make distcheck
   :
=======================================================
tutorial-program-1.0 archives ready for distribution:
tutorial-program-1.0.tar.gz
=======================================================


[frey@login00.darwin build-intel]$ mkdir X ; cd X
[frey@login00.darwin X]$ tar -xf ../tutorial-program-1.0.tar.gz
[frey@login00.darwin X]$ ls -l tutorial-program-1.0
[frey@login00.darwin X]$ ls -l tutorial-program-1.0
total 293
-rw-r--r-- 1 frey everyone  37794 Oct 14 15:05 aclocal.m4
-rw-r--r-- 1 frey everyone      0 Oct 14 15:04 AUTHORS
-rw-r--r-- 1 frey everyone      0 Oct 14 15:04 ChangeLog
-rw-r--r-- 1 frey everyone    625 Oct 14 15:05 config.h.in
-rwxr-xr-x 1 frey everyone 141680 Oct 14 15:05 configure
-rw-r--r-- 1 frey everyone    129 Sep 25 17:32 configure.ac
-rw-r--r-- 1 frey everyone  35147 Oct 14 15:05 COPYING
-rwxr-xr-x 1 frey everyone  23566 Oct 14 15:05 depcomp
-rw-r--r-- 1 frey everyone  15749 Oct 14 15:05 INSTALL
   :
```


- The "distcheck" produces a distributable software package with the configure script and all infrastructure files

GNU autotools

- This barely touches on the scope of autoconf
  - Many resources online — not all of them easy to understand
- Main caveat is portability
  - 100% tied to `make` build system
  - Originated on GNU Linux systems
  - Has been ported to other operating systems, but compatibility varies between releases
    - As changes are made to an OS, older autoconf tools may no longer work properly
  - Changes to compilers, new compilers, must be explicitly handled
    - Newer releases expand coverage of new, sometimes eliminate coverage of old

CMake

CMake

- To address the issues with autoconf
  - Use a simpler, more easily-understood language to define build environment
  - Allow for build systems other than just `make`
  - Portable between Linux/Mac/Windows with usage held in common

CMake

- To address the issues with autoconf
- Build system description created in `CMakeLists.txt` files

```
[frey@login00.darwin src-3]$ cd ../../src-4


[frey@login00.darwin src-4]$ ls -l
total 39
-rw-r--r-- 1 frey everyone  295 Sep 25 17:32 CMakeLists.txt
-rw-r--r-- 1 frey everyone  113 Sep 25 17:32 my_program.c
-rw-r--r-- 1 frey everyone  195 Sep 25 17:32 printargv.c
-rw-r--r-- 1 frey everyone  105 Sep 25 17:32 printargv.h
-rw-r--r-- 1 frey everyone 1423 Sep 25 17:32 README.md


[frey@login00.darwin src-4]$ cat CMakeLists.txt
cmake_minimum_required (VERSION 2.6)

# Define the project:
project (my_program C)

# The project includes an executable program:
add_executable(my_program my_program.c printargv.c)

# Installation should copy the executable into a "bin" directory:
install (TARGETS my_program DESTINATION bin)
```

CMake

- To address the issues with autoconf
- Build system description created in `CMakeLists.txt` files

```
[frey@login00.darwin src-3]$ cd ../../src-4


[frey@login00.darwin src-4]$ ls -l
total 39
-rw-r--r-- 1 frey everyone  295 Sep 25 17:32 CMakeLists.txt
-rw-r--r-- 1 frey everyone  113 Sep 25 17:32 my_program.c
-rw-r--r-- 1 frey everyone  195 Sep 25 17:32 printargv.c
-rw-r--r-- 1 frey everyone  105 Sep 25 17:32 printargv.h
-rw-r--r-- 1 frey everyone 1423 Sep 25 17:32 README.md


[frey@login00.darwin src-4]$ cat CMakeLists.txt
cmake_minimum_required (VE

# Define the project:
project (my_program C)

# The project includes an executable program:
add_executable(my_program my_program.c printargv.c)

# Installation should copy the executable into a "bin" directory:
install (TARGETS my_program DESTINATION bin)
```

The project is named "my_program" and uses the C language

CMake

- To address the issues with autoconf
- Build system description created in `CMakeLists.txt` files

```
[frey@login00.darwin src-3]$ cd ../../src-4


[frey@login00.darwin src-4]$ ls -l
total 39
-rw-r--r-- 1 frey everyone  295 Sep 25 17:32 CMakeLists.txt
-rw-r--r-- 1 frey everyone  113 Sep 25 17:32 my_program.c
-rw-r--r-- 1 frey everyone  195 Sep 25 17:32 printargv.c
-rw-r--r-- 1 frey everyone  105 Sep 25 17:32 printargv.h
-rw-r--r-- 1 frey everyone 1423 Sep 25 17:32 README.md


[frey@login00.darwin src-4]$ cat CMakeLists.txt
cmake_minimum_required (VERSION 2.6)

# Define the project:
project (my_program C)

# The project includes an executable program:
add_executable(my_program my_program.c printargv.c)

# Installation should copy the executable into a "bin" directory:
install (TARGETS my_program DESTINATION bin)
```

An executable target named "my_program" has ingredients "my_program.c" and "printargv.c"

CMake

- To address the issues with autoconf
- Build system description created in `CMakeLists.txt` files

```
[frey@login00.darwin src-3]$ cd ../../src-4


[frey@login00.darwin src-4]$ ls -l
total 39
-rw-r--r-- 1 frey everyone  295 Sep 25 17:32 CMakeLists.txt
-rw-r--r-- 1 frey everyone  113 Sep 25 17:32 my_program.c
-rw-r--r-- 1 frey everyone  195 Sep 25 17:32 printargv.c
-rw-r--r-- 1 frey everyone  105 Sep 25 17:32 printargv.h
-rw-r--r-- 1 frey everyone 1423 Sep 25 17:32 README.md


[frey@login00.darwin src-4]$ cat CMakeLists.txt
cmake_minimum_required (VERSION 2.6)

# Define the project:
project (my_program C)

# The project includes an executable program:
add_executable(my_program my_program.c printargv

# Installation should copy the executable into a
install (TARGETS my_program DESTINATION bin)
```

The "my_program" target is installed by copying its products into a directory named "bin" under the installation prefix directory

CMake

- To address the issues with autoconf
- Build system description created in `CMakeLists.txt` files

```
[frey@login00.darwin src-4]$ mkdir build-system ; cd build-system
[frey@login00.darwin build-system]$ vpkg_require cmake/default
Adding package `cmake/3.21.4` to your environment

[frey@login00.darwin build-system]$ CC=gcc cmake -DCMAKE_INSTALL_PREFIX=/tmp/abc ..
   :
-- The C compiler identification is GNU 4.8.5
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/1001/sw/unix-software-dev/src-4/build-system

[frey@login00.darwin build-system]$ make
[ 33%] Building C object CMakeFiles/my_program.dir/my_program.c.o
[ 66%] Building C object CMakeFiles/my_program.dir/printargv.c.o
[100%] Linking C executable my_program
[100%] Built target my_program
```


- Interesting naming scheme — don't drop the original extension (.c) but append the object extension to it (.c.o)
    - Preserves some sense of that language the original source was!

CMake

- To address the issues with autoconf
- Build system description created in `CMakeLists.txt` files

```
[frey@login00.darwin build-system]$ rm -rf * ; CC=gcc ccmake ..
```


•

CMake

- To address the issues
- Build system descript

```
[frey@login00.darwin build-system]$ rm -rf * ; CC=g
```

```
                                                   Page 1 of 1
 CMAKE_BUILD_TYPE                *
 CMAKE_INSTALL_PREFIX            */usr/local

CMAKE_BUILD_TYPE: Choose the type of build, options are: None Debug Release RelWithDebInfo MinSizeRe
Keys: [enter] Edit an entry [d] Delete an entry                              CMake Version 3.21.4
      [l] Show log output   [c] Configure
      [h] Help              [q] Quit without generating
      [t] Toggle advanced mode (currently off)
```


- Build type is used to determine what flags to pass to the compiler for optimization

CMake

- To address the issues with autoconf
- Build system description created in `CMakeLists.txt` files

```
#
# CMAKE_BUILD_TYPE not set:
#
[frey@login00.darwin build-system]$ make VERBOSE=1 2>&1 | grep gcc
/usr/bin/gcc     -MD -MT CMakeFiles/my_program.dir/my_program.c.o -MF CMakeFiles/my_program.dir/my_program.c.o.d -o CMakeFiles/my_program.dir/my_program.c.o -c
/home/1001/sw/unix-software-dev/src-4/my_program.c
/usr/bin/gcc     -MD -MT CMakeFiles/my_program.dir/printargv.c.o -MF CMakeFiles/my_program.dir/printargv.c.o.d -o CMakeFiles/my_program.dir/printargv.c.o -c
/home/1001/sw/unix-software-dev/src-4/printargv.c
/usr/bin/gcc -rdynamic CMakeFiles/my_program.dir/my_program.c.o CMakeFiles/my_program.dir/printargv.c.o -o my_program


#
# CMAKE_BUILD_TYPE set to Release
#
[frey@login00.darwin build-system]$ CC=gcc cmake -DCMAKE_BUILD_TYPE=Release ..
[frey@login00.darwin build-system]$ make VERBOSE=1 2>&1 | grep gcc
/usr/bin/gcc   -O3 -DNDEBUG -MD -MT CMakeFiles/my_program.dir/my_program.c.o -MF CMakeFiles/my_program.dir/my_program.c.o.d -o
CMakeFiles/my_program.dir/my_program.c.o -c /home/1001/sw/unix-software-dev/src-4/my_program.c
/usr/bin/gcc   -O3 -DNDEBUG -MD -MT CMakeFiles/my_program.dir/printargv.c.o -MF CMakeFiles/my_program.dir/printargv.c.o.d -o
CMakeFiles/my_program.dir/printargv.c.o -c /home/1001/sw/unix-software-dev/src-4/printargv.c
/usr/bin/gcc -O3 -DNDEBUG -rdynamic CMakeFiles/my_program.dir/my_program.c.o CMakeFiles/my_program.dir/printargv.c.o -o my_program
```


-

CMake

- To address the issues with autoconf
- Build system description created in `CMakeLists.txt` files

```
#
# CMAKE_BUILD_TYPE not set:
#
[frey@login00.darwin build-system]$ make VERBOSE=1 2>&1 | grep gcc
/usr/bin/gcc     -MD -MT CMakeFiles/my_program.dir/my_program.c.o -MF CMakeFiles/my_program.dir/my_program.c.o.d -o CMakeFiles/my_program.dir/my_program.c.o -c
/home/1001/sw/unix-software-dev/src-4/my_program.c
/usr/bin/gcc     -MD -MT CMakeFiles/my_program.dir/printargv.c.o -MF CMakeFiles/my_program.dir/printargv.c.o.d -o CMakeFiles/my_program.dir/printargv.c.o -c
/home/1001/sw/unix-software-dev/src-4/printargv.c
/usr/bin/gcc -rdynamic CMakeFiles/my_program.dir/my_program.c.o CMakeFiles/my_program.dir/printargv.c.o -o my_program


#
# CMAKE_BUILD_TYPE set to Release
#
[frey@login00.darwin build-system]$ CC=gcc cmake -DCMAKE_BUILD_TYPE=Release ..
[frey@login00.darwin build-system]$ make VERBOSE=1 2>&1 | grep gcc
/usr/bin/gcc   -O3 -DNDEBUG -MD -MT CMakeFiles/my_program.dir/my_program.c.o -MF CMakeFiles/my_program.dir/my_program.c.o.d -o
CMakeFiles/my_program.dir/my_program.c.o -c /home/1001/sw/unix-software-dev/src-4/my_program.c
/usr/bin/gcc   -O3 -DNDEBUG -MD -MT CMakeFiles/my_program.dir/printargv.c.o -MF CMakeFiles/my_program.dir/printargv.c.o.d -o
CMakeFiles/my_program.dir/printargv.c.o -c /home/1001/sw/unix-software-dev/src-4/printargv.c
/usr/bin/gcc -O3 -DNDEBUG -rdynamic CMakeFiles/my_program.dir/my_program.c.o CMakeFiles/my_program.dir/printargv.c.o -o my_program
```


- Recall the "MD" flag was used to emit dependency rules for the sake of make
  - The "MF" option specifies to what file the rules should be written

CMake

- To address the issues with autoconf
- Build system description created in `CMakeLists.txt` files

```
[frey@login00.darwin build-system]$ rm -rf *
[frey@login00.darwin build-system]$ CC=gcc cmake -DCMAKE_INSTALL_PREFIX:PATH=/tmp/def -DCMAKE_BUILD_TYPE=Release ..
   :


[frey@login00.darwin build-system]$ make install
Scanning dependencies of target my_program
[ 50%] Building C object CMakeFiles/my_program.dir/my_program.c.o
[100%] Building C object CMakeFiles/my_program.dir/printargv.c.o
Linking C executable my_program
[100%] Built target my_program
Install the project...
-- Install configuration: "Release"
-- Installing: /tmp/def/bin/my_program


[frey@login00.darwin build-system]$ /tmp/def/bin/my_program $((4*10)) b c
1       40
2       b
3       c
```


-

CMake

- Multi-directory projects
  - Each directory gets a CMakeLists.txt file defining the build therein
  - Parent directory CMakeLists.txt must reference the sub-directory with the `add_subdirectory()` command
  - Example 5 implements a multi-directory project in CMake v3.x versus the older v2.x language as in Example 4

CMake: Example 5

1. Top-level CMakeLists.txt
   a. **All** compile statements should reference the libprintargv directory for header search
   b. Both source subdirectories added

```
cmake_minimum_required(VERSION 3.20)

# Define the project, languages used, version, etc.
project(my_program
        VERSION 0.0.1
        DESCRIPTION "Demonstration program build with CMake"
        LANGUAGES C
        )
include(GNUInstallDirs)

# By default build shared libraries without rpath embedded when they're inst
option(BUILD_SHARED_LIBS "Build shared libraries, not static" ON)

# Set if not already specified:
if (NOT DEFINED CMAKE_SKIP_INSTALL_RPATH)
        set(CMAKE_SKIP_INSTALL_RPATH TRUE CACHE BOOL "Do not embed rpaths i
endif ()

# Be sure the headers in the libprintargv sub-project are visible:
include_directories (${PROJECT_SOURCE_DIR})

# Add the sub-projects:
add_subdirectory (libprintargv)
add_subdirectory (my_program)
```


•

CMake: Example 5

1. Top-level CMakeLists.txt
2. libprintargv
   a. Create a library target, shared or static based on value of BUILD_SHARED_LIBS variable
   b. Associate the header file with the target (for the sake of installation)
   c. Library install destination may vary by system (lib, lib64)

```
# Produce a library (note we don't include the "lib" prefix that Unix/Linux
# typically wants or the type extension ".so" or ".a" -- this is a base name
# and CMake will figure out what prefix/suffix is necessary for the OS in
# question):
if (${BUILD_SHARED_LIBS})
        add_library(printargv SHARED printargv.c)
else ()
        add_library(printargv printargv.c)
endif ()

# Associate header files with the library target:
set_target_properties(printargv PROPERTIES PUBLIC_HEADER "printargv.h")

# Installation should copy the library to the "lib" directory and public
# header files to the "include" directory:
install (
        TARGETS printargv DESTINATION ${CMAKE_INSTALL_LIBDIR}
        PUBLIC_HEADER DESTINATION ${CMAKE_INSTALL_INCLUDEDIR})
```


-

CMake: Example 5

1. Top-level CMakeLists.txt
2. libprintargv
3. my_program
   a. Create an executable program
   b. Link the program against the printargv target (defined in libprintargv)

```
# Build an executable:
add_executable(my_program my_program.c)

# The "my_program" target must be linked against the library in
# the "printargv" target that was defined in the libprintargv
# directory:
target_link_libraries (my_program printargv)

# Set the embedded rpath to the installation's library
# directory:
if ( NOT ${CMAKE_SKIP_INSTALL_RPATH} )
    set_target_properties(my_program PROPERTIES
        INSTALL_RPATH ${CMAKE_INSTALL_FULL_LIBDIR})
endif ()

# Installation should copy the executable to the "bin" directory:
install (TARGETS my_program DESTINATION ${CMAKE_INSTALL_BINDIR})
```


•

CMake: Example 5

- Build optimized with GCC 12.2, install to /tmp/abc:

```
[frey@login00.darwin ~]$ cd ~/sw/unix-software-dev/src-5

[frey@login00.darwin src-5]$ mkdir build-gcc-12.2 ; cd build-gcc-12.2

[frey@login00.darwin build-gcc-12.2]$ vpkg_require cmake/default gcc/12.2
Adding package `cmake/3.21.4` to your environment
Adding package `gcc/12.2.0` to your environment


[frey@login00.darwin build-gcc-12.2]$ CC=$(which gcc) cmake -DCMAKE_INSTALL_PREFIX=/tmp/abc -DCMAKE_BUILD_TYPE=Release ..
-- The C compiler identification is GNU 12.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /opt/shared/gcc/12.2.0/bin/gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/1001/sw/unix-software-dev/src-5/build-gcc-12.2
```

CMake: Example 5

- Build optimized with GCC 12.2, install to /tmp/abc:

```
[frey@login00.darwin build-gcc-12.2]$ make
[ 25%] Building C object libprintargv/CMakeFiles/printargv.dir/printargv.c.o
[ 50%] Linking C shared library libprintargv.so
[ 50%] Built target printargv
[ 75%] Building C object my_program/CMakeFiles/my_program.dir/my_program.c.o
[100%] Linking C executable my_program
[100%] Built target my_program


[frey@login00.darwin build-gcc-12.2]$ ls -l
total 58
-rw-r--r-- 1 frey everyone 15827 Oct 15 11:22 CMakeCache.txt
drwxr-xr-x 4 frey everyone    11 Oct 15 11:24 CMakeFiles
-rw-r--r-- 1 frey everyone  2050 Oct 15 11:22 cmake_install.cmake
drwxr-xr-x 3 frey everyone     6 Oct 15 11:24 libprintargv
-rw-r--r-- 1 frey everyone  7262 Oct 15 11:22 Makefile
drwxr-xr-x 3 frey everyone     6 Oct 15 11:24 my_program


[frey@login00.darwin build-gcc-12.2]$ ./my_program/my_program {a,b}{0,1}
1       a0
2       a1
3       b0
4       b1
```


-

CMake: Example 5

- Build optimized with GCC 12.2, install to /tmp/abc:

```
[frey@login00.darwin build-gcc-12.2]$ make install
Consolidate compiler generated dependencies of target printargv
[ 50%] Built target printargv
Consolidate compiler generated dependencies of target my_program
[100%] Built target my_program
Install the project...
-- Install configuration: "Release"
-- Installing: /tmp/abc/lib64/libprintargv.so
-- Installing: /tmp/abc/include/printargv.h
-- Installing: /tmp/abc/bin/my_program
-- Set runtime path of "/tmp/abc/bin/my_program" to "/tmp/abc/lib64"


[frey@login00.darwin build-gcc-12.2]$ find /tmp/abc
/tmp/abc
/tmp/abc/lib64
/tmp/abc/lib64/libprintargv.so
/tmp/abc/include
/tmp/abc/include/printargv.h
/tmp/abc/bin
/tmp/abc/bin/my_program
```

CMake: Example 5

- Alternative: do **not** embed "rpath" in the `my_program` executable

```
[frey@login00.darwin build-gcc-12.2]$ rm -rf *


[frey@login00.darwin build-gcc-12.2]$ CC=$(which gcc) cmake -DCMAKE_INSTALL_PREFIX=/tmp/abc -DCMAKE_BUILD_TYPE=Release \
                                                            -DCMAKE_SKIP_INSTALL_RPATH=On ..

[frey@login00.darwin build-gcc-12.2]$ make install
[ 25%] Building C object libprintargv/CMakeFiles/printargv.dir/printargv.c.o
[ 50%] Linking C shared library libprintargv.so
[ 50%] Built target printargv
[ 75%] Building C object my_program/CMakeFiles/my_program.dir/my_program.c.o
[100%] Linking C executable my_program
[100%] Built target my_program
Install the project...
-- Install configuration: "Release"
-- Installing: /tmp/abc/lib64/libprintargv.so
-- Up-to-date: /tmp/abc/include/printargv.h
-- Installing: /tmp/abc/bin/my_program
-- Set runtime path of "/tmp/abc/bin/my_program" to ""


[frey@login00.darwin build-gcc-12.2]$ /tmp/abc/bin/my_program {a,b}{0,1}
/tmp/abc/bin/my_program: error while loading shared libraries: libprintargv.so: cannot open shared object file: No such file or
director
```


-

CMake: Example 5

- Alternative: do **not** embed "rpath" in the `my_program` executable

```
[frey@login00.darwin build-gcc-12.2]$ find /tmp/abc
/tmp/abc
/tmp/abc/lib64
/tmp/abc/lib64/libprintargv.so
/tmp/abc/include
/tmp/abc/include/printargv.h
/tmp/abc/bin
/tmp/abc/bin/my_program


[frey@login00.darwin build-gcc-12.2]$ LD_LIBRARY_PATH=/tmp/abc/lib64 /tmp/abc/bin/my_program {a,b}{0,1}
1       a0
2       a1
3       b0
4       b1
```

CMake: Example 5

- Alternative: rebuild with debug enabled, use debug library at runtime

```
[frey@login00.darwin build-gcc-12.2]$ rm -rf *


[frey@login00.darwin build-gcc-12.2]$ CC=$(which gcc) cmake -DCMAKE_INSTALL_PREFIX=/tmp/abc -DCMAKE_BUILD_TYPE=Debug \
                                                           -DCMAKE_SKIP_INSTALL_RPATH=On ..
   :
[frey@login00.darwin build-gcc-12.2]$ make
[ 25%] Building C object libprintargv/CMakeFiles/printargv.dir/printargv.c.o
[ 50%] Linking C shared library libprintargv.so
[ 50%] Built target printargv
[ 75%] Building C object my_program/CMakeFiles/my_program.dir/my_program.c.o
[100%] Linking C executable my_program
[100%] Built target my_program


[frey@login00.darwin build-gcc-12.2]$ export PATH="/tmp/abc/bin:$PATH"
[frey@login00.darwin build-gcc-12.2]$ LD_LIBRARY_PATH="$(pwd)/libprintargv" my_program {a,b}{0,1}
DEBUG: enter printargv
1       a0
2       a1
3       b0
4       b1
DEBUG: exit printargv
```


- The executable in /tmp/abc is still the optimized build, but the shared library is a debug build

CMake: Example 5

- Alternative: override embedded library search path at runtime

```
[frey@login00.darwin build-gcc-12.2]$ cp ./libprintargv/libprintargv.so /tmp/abc/lib64/libprintargv-debug.so

#
# Rebuild and reinstall with /tmp/abc/lib64 embedded in the executable...
#

[frey@login00.darwin build-gcc-12.2]$ my_program {a,b}{0,1}
1       a0
2       a1
3       b0
4       b1


[frey@login00.darwin build-gcc-12.2]$ LD_PRELOAD=/tmp/abc/lib64/libprintargv-debug.so my_program {a,b}{0,1}
DEBUG: enter printargv
1       a0
2       a1
3       b0
4       b1
DEBUG: exit printargv
```


- Pre-loading brings all of the cited libraries' symbols into memory so that after the executable is loaded NONE are undefined — which would trigger the library search procedure
- Special debugging libraries, for memory management operations for example, are often loaded this way

Runtime Environment

Runtime Environment

- In Example 5 several environment variables were leveraged
  - PATH: colon-separated list of directories to search for executables
  - LD_LIBRARY_PATH: colon-separated list of directories to search for shared libraries
  - Others (e.g. MANPATH) provide other search paths

Runtime Environment

- In Example 5 several environment variables were leveraged
  - PATH: colon-separated list of directories to search for executables
  - LD_LIBRARY_PATH: colon-separated list of directories to search for shared libraries
  - Others (e.g. MANPATH) provide other search paths
- Compiled *object code* is assembled into an executable by a *linker*
  - Resolves named objects (functions, variables) to addresses
  - Addresses may be present in the executable OR a reference to a shared library

Runtime Environment

- In Example 5 several environment variables were leveraged
  - PATH: colon-separated list of directories to search for executables
  - LD_LIBRARY_PATH: colon-separated list of directories to search for shared libraries
  - Others (e.g. MANPATH) provide other search paths
- Compiled *object code* is assembled into an executable by a *linker*
  - Resolves named objects (functions, variables) to addresses
  - Addresses may be present in the executable OR a reference to a shared library
- When the program is executed, references to shared libraries are resolved by the *runtime linker*
  - Embedded rpath(s) checked
  - Paths in LD_LIBRARY_PATH checked

Runtime Environment

- Default paths for installed software components in Linux/Unix
  - `bin`: user-accessible executables
  - `sbin`: privileged user-accessible executables
  - `lib`, `lib64`: shared libraries
  - `libexec`: helper programs/libraries associated with an executable or library
  - `man`, `share/man`: man pages
  - `include`: header files (for software development)
- autoconf and CMake often leverage the same hierarchy for installs
  - Clear categorization of components in both OS and third-party software
  - Default Linux install prefixes:
    - OS = `/usr`
    - Add-ons = `/usr/local`

Runtime Environment

- Default paths for installed software components in Linux/Unix
  - `bin`: user-accessible executables
  - `sbin`: privileged user-accessible executables
  - `lib`, `lib64`: shared libraries
  - `libexec`: helper programs/libraries associated with an executable or library
  - `man`, `share/man`: man pages
  - `include`: header files (for software development)
- autoconf and CMake often leverage the same hierarchy for installs
  - Clear categorization of components in both OS and third-party software
  - Default Linux install prefixes:
    - OS = `/usr`
    - Add-ons = `/usr/local`

How can we install multiple versions/variants of the `my_program` executable in `/usr/local`??

UD IT Research Cyberinfrastructure

- Add a suffix to the my_program name, such as my_program — hyphen — v1.0
  - The same suffixing must be used on shared libraries accompanying the program so they can be versioned

Runtime Environment

- Naming suffixes may not be supported by all software build environments
  - Our projects do not support adding `-v1.0` to the executable, library, and header names
- Isolate a version/variant of the software in its own *installation root*
  - The `--prefix` option in autoconf; `CMAKE_INSTALL_PATH` in CMake
  - On a Linux system, the components are separated into the usual component paths
    - `bin`, `lib`, et al.
- Use a versioned-package hierarchy:
  - `<software-hierarchy>/<package-name>/<version-or-variant-name>`
  - E.g. `/opt/shared/openmpi/4.1.5-gcc-12.2`
    - `<software-hierarchy> = /opt/shared`
    - `<package-name> = openmpi`
    - `<version-or-variant-name> = 4.1.5-gcc-12.2`

Runtime Environment

- Where will components be found?
  - User executables: `/opt/shared/openmpi/4.1.5-gcc-12.2/bin`
  - Libraries: `/opt/shared/openmpi/4.1.5-gcc-12.2/lib`
  - Man pages: `/opt/shared/openmpi/4.1.5-gcc-12.2/share/man`
  - Development headers: `/opt/shared/openmpi/4.1.5-gcc-12.2/include`
- To use this software, environment variables must be augmented
  - Add respective directories to `PATH`, `LD_LIBRARY_PATH`, `MANPATH`
  - For the sake of development (autoconf, CMake, and others)
    - Add "`-I/opt/shared/openmpi/4.1.5-gcc-12.2/include`" to `CPPFLAGS`
    - Add "`-L/opt/shared/openmpi/4.1.5-gcc-12.2/lib`" to `LDFLAGS`

Runtime Environment

- Where will components be found?
  - User executables: `/opt/shared/openmpi/4.1.5-gcc-12.2/bin`
  - Libraries: `/opt/shared/openmpi/4.1.5-gcc-12.2/lib`
  - Man pages: `/opt/shared/openmpi/4.1.5-gcc-12.2/share/man`
  - De
- To use
  - Ad
  - For
    -
    -

Many software products will suggest these environment variable changes or other environment alterations should be added to the user's `~/.bashrc` or `~/.bash_login` file, for example.

Do not follow this advice!

Runtime Environment

- Why not alter shell login files?
  - Whatever is present in the shell login files affects **every shell** on the cluster
    - Not just your login shell — every job shell, too
  - `PATH=/home/1001/sw/pkg/2/bin:/home/1001/sw/pkg/1/bin:…`
    - Once you've added version 2, how do you selectively reference version 1?
  - `LD_LIBRARY_PATH=/home/1001/sw/pkg/2/lib:/home/1001/sw/other_pkg/1/lib:…`
    - If same library is present in both — with e.g. alternative features — the runtime linker may not find the correct copy
  - Python virtual environments (virtualenvs) are particularly bad
    - Many will contain libraries and executables that override what the OS provides
    - In some cases an entire development runtime is embedded (e.g. Ubuntu-style GNU libraries)
      - OS commands may not work with the overriding library versions

Runtime Environment

- Why not alter shell login files?
  - Whatever is present in the shell login files affects **every shell** on the cluster
    - Not just your login shell — every job shell, too
  - `PATH=/home/1001/sw/pkg/2/bin:/home/1001/sw/pkg/1/bin:…`
    - ■ … 1?
  - `LD_` … `:…`
    - ■ … untime linker
  - Pyt…
    - ■ … rovides
    - ■ … u-style GNU
      - OS commands may not work with the overriding library versions

What's the alternative?

Keep *all* shells as minimally-altered as possible and make changes on-demand, only as necessary to the situation.

When you need to use `/home/1001/sw/pkg/1`, add its directories to `PATH`, `LD_LIBRARY_PATH`, etc.

VALET

VALET

- Similar to the popular *modules* environment management
- A *package definition file* describes environment alterations associated with one or more *versions/variants* of a *package*
    - Additions to `PATH`, `LD_LIBRARY_PATH`, `MANPATH`
    - Changes to arbitrary environment variables
    - Define command aliases
    - Execute (source) shell scripts
    - Load other packages — *dependencies* — into the environment, too

VALET

- Similar to the popular *modules* environment management
- A *package definition file* describes environment alterations associated with one or more *versions/variants* of a *package*
- How are package definition files found?
    - Default paths:
        - IT RCI managed software: `/opt/shared/valet/etc`
        - User managed: `~/.valet`
    - Value of `VALET_PATH` behaves like `PATH`, `LD_LIBRARY_PATH`
        - Added by `workgroup`: `${WORKDIR}/sw/valet` (if the path exists)
        - Users can add their own paths
            - `export VALET_PATH="/home/1001/sw/valet:$VALET_PATH"`

- By "variants" I mean multiple copies of the same version with different build options — like compiler choice — or with alternative parameterizations

VALET

- How are package definition files created?
  - YAML (YAhoo Markup Language) is the preferred format
    - JSON (JavaScript Object Notation) and XML (eXtensible Markup Language) available but not recommended
  - File should be named `<pkg-id>.vpkg_yaml`
  - Document is structured with default actions and a dictionary of per-version actions
  - Example presented here is in `unix-software-dev/valet-2.1/yaml`

VALET

- Package-level details
  - The `prefix` is the directory that contains all versions/variants of the package
  - The `actions` is a list of alterations to apply to the environment
    - Treat `DUMMY_CONF_PATH` as a colon-delimited list (like `PATH`) and add a value to the list
    - The value to add is the `etc` directory inside the directory containing the version/variant of the package

```
dummy:
    description: Dummy - The software without source or purpose
    url: http://www.udel.edu/dummy-software/
    prefix: /home/1001/sw/dummy
    actions:
        - variable: DUMMY_CONF_PATH
          action: append-path
          value: "${VALET_PATH_PREFIX}/etc"
```


- No tabbed whitespace is allowed in YAML, only regular space characters

VALET

- Version-level details
  - A version `prefix` directory can be specified explicitly **or** it is implied — the *version id* is appended to the package's `prefix` directory
    - `/home/1001/sw/dummy/0.1`
  - The `dependencies` is a list of other versioned package ids that are required by this versioned package
    - The dummy program was compiled with GCC 12.2

```yaml
dummy:
    description: Dummy - The software without source or purpose
    url: http://www.udel.edu/dummy-software/
    prefix: /home/1001/sw/dummy
    actions:
        - variable: DUMMY_CONF_PATH
          action: append-path
          value: "${VALET_PATH_PREFIX}/etc"

    versions:
        0.1:
            description: Version 0.1 (compiled with GCC 12)
            actions:
                - variable: DUMMY_CC
                  action: set
                  value: gcc
            dependencies:
                - gcc/12.2
```


-

VALET

- A default version can be explicit or implied
- A version id can have *feature* tags
  - Comma-separated list of strings occurring after a colon
  - Unordered: "a,b,c" == "b,c,a" == "c,a,b"
  - Implicit directory name includes the tags: e.g. 0.1-intel
- A version can be an alternate name for another version id
  - an alias

```
dummy:
    description: Dummy - The software without source or purpose
    url: http://www.udel.edu/dummy-software/
    prefix: /home/1001/sw/dummy
    actions:
        - variable: DUMMY_CONF_PATH
          action: append-path
          value: "${VALET_PATH_PREFIX}/etc"
    default-version: 0.1
    versions:
        0.1:
            description: Version 0.1 (compiled with GCC 12)
            actions:
                - variable: DUMMY_CC
                  action: set
                  value: gcc
            dependencies:
                - gcc/12.2
        0.1:intel:
            description: Version 0.1 (compiled with Intel 2020)
            actions:
                - variable: DUMMY_CC
                  action: set
                  value: icc
            dependencies:
                - intel/2020
        intel:
            alias-to: 0.1:intel
```


-

VALET

- "I don't see any mention of a `bin` directory to add to `PATH`…"
  - VALET automatically checks for standard Linux component directories and adds them to the appropriate environment variables
    - executables, libraries, man pages, etc.
  - Other paths can be explicitly specified
    - Absolute paths
    - Paths relative to the version's `prefix` directory

```
dummy:
    description: Dummy - The software without source or purpose
    url: http://www.udel.edu/dummy-software/
    prefix: /home/1001/sw/dummy
    actions:
        - variable: DUMMY_CONF_PATH
          action: append-path
          value: "${VALET_PATH_PREFIX}/etc"
    default-version: 0.1
    versions:
        0.1:
            description: Version 0.1 (compiled with GCC 12)
            actions:
                - variable: DUMMY_CC
                  action: set
                  value: gcc
            dependencies:
                - gcc/12.2
        0.1:intel:
            description: Version 0.1 (compiled with Intel 2020)
            actions:
                - variable: DUMMY_CC
                  action: set
                  value: icc
                - bindir: build/exe
                - libdir: build/libs
            dependencies:
                - intel/2020
        intel:
            alias-to: 0.1:intel
```


•

VALET

- "I don't see any mention of a `bin` directory to add to `PATH`."
  - The automatic detection of standard component directories means many package definitions can be relatively simple — like `pmix` shown here.
    - pages, etc.
  - Other paths can be explicitly specified
    - Absolute paths
    - Paths relative to the version's `prefix` directory

```
[frey@login01.darwin ~]$ cat /opt/shared/valet/etc/pmix.vpkg_yaml
pmix:
    prefix: /opt/shared/pmix
    description: "PMIx is an application programming interface (API) standar
    url: "https://github.com/pmix/pmix-standard"

    default: "3"

    versions:
        "4":
            alias-to: "4.1.2"
        "4.1.2":
            description: system compilers, Lustre, Omni-path, Slurm

        "3":
            alias-to: "3.2.1"
        "3.2.1":
            description: system compilers, Slurm
```

```
            - variable: DUMMY_CC
              action: set
              value: icc
            - bindir: build/exe
            - libdir: build/libs
          dependencies:
            - intel/2020
      intel:
          alias-to: 0.1:intel
```


- It can be very useful to examine the package definition files that IT RCI maintains to gain a better understanding of VALET

VALET

- "How do I check if my package definition is valid?"
  - The `vpkg_check` command validates the file and prints the parsed definition if successful
  - If not successful, information is provided re: at which line the error occurred or what convention was violated
  - Note the implicit `prefix` directories are displayed
  - Copy/move the file to e.g. `~/.valet` to install

```
[frey@login01.darwin yaml]$ vpkg_check dummy.vpkg_yaml
dummy.vpkg_yaml is OK

[dummy] {
  contexts: all
  actions: {
    DUMMY_CONF_PATH=+[:]${VALET_PATH_PREFIX}/etc (contexts: all)
    DUMMY_PREFIX=${VALET_PATH_PREFIX} (contexts: development)
  }
  http://www.udel.edu/dummy-software/
  Dummy - The software without source or purpose
  prefix: /home/1001/sw/dummy
  source file: /home/1001/sw/unix-software-dev/valet-2.1/yaml/dummy.vpkg_yam
  default version: dummy/0.1
  versions: {
    [dummy/0.1] {
      contexts: all
      actions: {
        DUMMY_CC=gcc (contexts: all)
      }
      dependencies: {
        gcc/12.2
      }
      Version 0.1 (compiled with GCC 12)
      prefix: /home/1001/sw/dummy/0.1
    }
    [dummy/0.1:intel] {
      contexts: all
      actions: {
        DUMMY_CC=icc (contexts: all)
        PATH+=[:]${VALET_PATH_PREFIX}/build/exe (contexts: all)
[frey@login01.darwin yaml]$cp dummy.vpkg_yaml ~/.valet
```


-

VALET

- "How do I check what packages are available?"
  - The `vpkg_list` command
  - Parsable package definitions in each (implicit and explicit) directory have their package id shown

```
[frey@login01.darwin ~]$ vpkg_list

Available packages:
  in /home/1001/.valet
    dummy
    dvipng
    libxc
    opencor
    vasp
  in /opt/shared/valet/2.1/etc
    alphafold
    amd-rocm
    amd-uprof
    anaconda
    aocl
    arpack
    atlas
    autoconf
    binutils
    blacs
    blis
    boost
    ccrypt
    charm
    cmake
    colabfold
    cryosparc
    cuda
    cudnn
    eigen
    emacs
    fftw
      :
```


•

VALET

- "How do I check what versions or variants of a package are available?"
  - The `vpkg_versions` command

```
[frey@login01.darwin ~]$ vpkg_versions dummy

Available versions in package (* = default version):

[/home/1001/.valet/dummy.vpkg_yaml]
dummy        Dummy - The software without source or purpose
* 0.1        Version 0.1 (compiled with GCC 12)
  0.1:intel  Version 0.1 (compiled with Intel 2020)
  intel      alias to dummy/0.1:intel
```


-

VALET

- "How do I add a versioned package to the environment?"
  - The `vpkg_require` command
  - The `vpkg_devrequire` command selects the *development* context
    - Additional changes are made to e.g. `CPPFLAGS`, `LDFLAGS`, etc.
- "How do I check what packages have already been added?"
  - The `vpkg_history` command

```
[frey@login01.darwin ~]$ vpkg_require dummy/intel
Adding dependency `intel/2020u4` to your environment
Adding package `dummy/0.1:intel` to your environment


[frey@login01.darwin ~]$ vpkg_history
[standard]
  intel/2020u4
  dummy/0.1:intel
```


•

VALET

- "How do I undo the changes made by `vpkg_require` or `vpkg_devrequire` commands?"
  - The `vpkg_rollback` command
    - No arguments = remove the last set of changes
    - Integer argument *<N>* = remove the last *<N>* sets of changes
    - "all" = remove all sets of changes

```
[frey@login01.darwin ~]$ vpkg_devrequire udunits/2.2
Adding package `udunits/2.2.28` to your environment

[frey@login01.darwin ~]$ vpkg_history
[standard]
  intel/2020u4
  dummy/0.1:intel
[development]
  udunits/2.2.28

[frey@login01.darwin ~]$ echo $CPPFLAGS
-I/opt/shared/udunits/2.2.28/include

[frey@login00.darwin ~]$ echo $UDUNITS_PREFIX
/opt/shared/udunits/2.2.28

[frey@login01.darwin ~]$ vpkg_rollback
[frey@login01.darwin ~]$ vpkg_history
[standard]
  intel/2020u4
  dummy/0.1:intel

[frey@login01.darwin ~]$ echo $CPPFLAGS

[frey@login01.darwin ~]$ vpkg_rollback all
[frey@login01.darwin ~]$ vpkg_history

[frey@login01.darwin ~]$
```


-

VALET

- "How do I undo the changes made by ... or

When using `vpkg_devrequire`, each package's `prefix` directory is added to an environment variable, `<pkg-id>_PREFIX`.

the last *<N>* sets of changes
  - "all" = remove all sets of changes

```
[frey@login01.darwin ~]$ vpkg_devrequire udunits/2.2
Adding package `udunits/2.2.28` to your environment

[frey@login01.darwin ~]$ vpkg_history
[standard]
  intel/2020u4
  dummy/0.1:intel
[development]
  udunits/2.2.28

[frey@login01.darwin ~]$ echo $CPPFLAGS
-I/opt/shared/udunits/2.2.28/include

[frey@login00.darwin ~]$ echo $UDUNITS_PREFIX
/opt/shared/udunits/2.2.28

[frey@login01.darwin ~]$ vpkg_rollback
[frey@login01.darwin ~]$ vpkg_history
[standard]
  intel/2020u4
  dummy/0.1:intel

[frey@login01.darwin ~]$ echo $CPPFLAGS

[frey@login01.darwin ~]$ vpkg_rollback all
[frey@login01.darwin ~]$ vpkg_history

[frey@login01.darwin ~]$
```


- The prefix variables are very useful in directing autotools and CMake where to find dependencies

VALET

- Multiple versioned package ids can be specified in a single command
  - A package's default version is selected with the `default` version id
  - Any conflicts or errors result in the entire set of changes' being rolled back (not applied)

```
[frey@login01.darwin ~]$ vpkg_require dummy/default valgrind/3.22.0 r/4.1.3
gcc/11.2.0 conflicts with gcc/12.2.0 already added to environment
```

VALET

- Multiple versioned package ids can be specified in a single command
  - A package's default version is selected with the `default` version id
  - Any conflicts or errors result in the entire set of changes' being rolled back (not applied)
    - Ignoring conflicts may produce runtime errors!
  - **It is always best-practice to minimize the number of sets of changes made to the environment.**

```
[frey@login01.darwin ~]$ vpkg_require dummy/default valgrind/3.22.0 r/4.1.3
gcc/11.2.0 conflicts with gcc/12.2.0 already added to environment


[frey@login01.darwin ~]$ vpkg_require --force \
>                               dummy/default valgrind/3.22.0 r/4.1.3
Adding dependency `gcc/12.2.0` to your environment
Adding package `dummy/0.1` to your environment
Adding package `valgrind/3.22.0` to your environment
Adding dependency `gcc/11.2.0` to your environment
Adding dependency `atlas/3.10.3` to your environment
Adding package `r/4.1.3` to your environment


[frey@login01.darwin ~]$ which R
/opt/shared/r/4.1.3/bin/R


[frey@login01.darwin ~]$ which valgrind
/opt/shared/valgrind/3.22.0/bin/valgrind
```


-

In Summary

In Summary

1. Adopt a hierarchical organization of software
   - `<base-dir>/<pkg-name>/<version-or-variant>`
   - `<base-dir>` can be anywhere, but common locations are:
     - `<base-dir> = ~/sw` (for personally-managed software)
     - `<base-dir> = ${WORKDIR}/sw` (for software shared by whole workgroup)

In Summary

1. Adopt a hierarchical organization of software
2. Adopt the Unix/Linux component directory layout
   - `bin`, `lib`, `include`, `share/man`, etc.
   - This often happens by default for software built via autotools or CMake

In Summary

1. Adopt a hierarchical organization of software
2. Adopt the Unix/Linux component directory layout
3. DO NOT install software by altering login files!
    - `.bashrc`, `.bash_profile` — hands off!

In Summary

1. Adopt a hierarchical organization of software
2. Adopt the Unix/Linux component directory layout
3. DO NOT install software by altering login files!
4. Use VALET to manage installed software
    - If you've followed 1. and 2. then this is much easier!

In Summary

1. Adopt a hierarchical organization of software
2. Adopt the Unix/Linux component directory layout
3. DO NOT install software by altering login files!
4. Use VALET to manage installed software
5. When developing software
    - Avoid writing *build scripts*: autotools, CMake, et al. will always do a better job
    - Relocatable: write software that has no fixed installation directory
    - Flexible: write software that is configurable at run time, not compile time
    - Batchable[1]: write software that requires **no** interactive input

[1] Yes, I made that word up. Copyright © Jeffrey Frey, 2024.

In Summary

1. Adopt a hierarchical organization of software
2. Adopt the Unix/Linux component directory layout
3. DO NOT install software by altering login files!
4. Use VALET to manage installed software
5. When developing software
6. Never be afraid to seek help…
    - …after you've exhausted other options: web search, documentation