

COSC 3750

Miscellaneous

Kim Buckner

University of Wyoming

Feb. 15, 2022

Last time

- Talked about argc and argv.
- Talked about wycat
- Questions?

Make

- One of the things people like to do is get tricky.
- Usually leads to problems.
- Make's “automatic” variables are a case in point.

(more ...)

\$@

- This is the filename of the target. The entire name.
- That means whatever is to the left of the colon.
- Great if you are lazy, also great for *pattern rules*.

(more ...)

\$ <

- The name of ONLY THE FIRST prerequisite.
- That is not all of them, be careful.

(more ...)

\$?

- The names of all the prerequisites NEWER than the target.
- Again, not useful for compiling.

(more ...)

\$^

- The names of ALL the prerequisites.
- DO NOT USE THIS if there are .h files in the prerequisites.
- DO PUT .h files in the prerequisites.

(more ...)

\$*

- The stem of the target in implicit and pattern rules.
- That is whatever would match the %.
- There is a but.

(more ...)

- If you have an explicit rule

`file.xyz : ...`

and “.xyz” is a recognized extension, then
\$* is “file.”

- If “.xyz” is not recognized, \$* is empty.
- “You should generally avoid using ‘\$*’ except in implicit rules or static pattern rules.”

Others you don't use

There are some very special versions of these.
An example is:

“‘\$(?D)’ and ‘\$(?F)’: Lists of the directory parts and the file-within-directory parts of all prerequisites that are newer than the target.”

You explain it to me.

Pattern rules.

```
%.o : %.c  
      ${CC} -c ${CFLAGS} ${CPPFLAGS} $<
```

```
%.tab.c %.tab.h: %.y  
      bison -d $<
```

```
% : %.c %.h  
      ${CC} ${CFLAGS} $< -o $@
```

(more ...)

- These are very handy when you have a large number of files that can be processed **exactly** the same way or you reuse *makefiles*.
- The % is a string that is the “stem” of the filename.
- You cannot change it from one side of the : to the other.

Rule errors

- Make halts on all errors by default.
- However, in cases such as “clean” rules, might not want that.
- If we precede the command with a ‘-’, the error does not cause make to quit.

`-rm *.o`

- The ‘-’ is NOT passed to shell with the command.

Noise

- Well not really.
- By default all commands are printed before they are executed.
- If you have say an *echo* command that is redundant.
- Precede the command with an at symbol to suppress the printing.

```
@echo ‘‘Now compiling blah’’
```

Just checking

- Suppose you want to know if there is any reason to “make” a target.
- But you do not want to really make it.
- Just use the “-n” option, as in

```
make -n
```

and you get a “dry-run.”
- The rules needed to update the file(s) will be printed but nothing done.

printf()

What is it?

- This function in various forms is for formatted output.
 - These only require `stdio.h`
 - `printf`, `fprintf`, `dprintf`, `sprintf`, `snprintf`
 - These also require `stdarg.h`
 - `vprintf`, `vfprintf`, `vdprintf`, `vsprintf`, `vsnprintf`

And so?

- printf writes to stdout.
- fprintf writes to the **FILE *** first argument.
- dprintf writes to a file descriptor
- sprintf and snprintf write to a **char *** first argument

(more ...)

- The “v” versions do not use individual arguments after the format string.
- They use a variable number of arguments, specifically a “va_list”.
- Basically, do not worry about them.

Normal operation

- The idea is that we have as the first argument (after `fd/FILE*/char*`) a string that contains formatting.
- That is followed by 0 or more arguments that are values to be used for the specified formats.

(more ...)

- Remember my showing you that -Wall will check the format against the arguments?
- You must make sure that you do not mix them up.
- Cannot always ensure that the gcc checks are all-inclusive.
- And it CANNOT check for '\0' at end of string.

Basics of the format

- There are a large number of format characters.
- You **MUST** read the man page.
- But many read them without reading carefully OR understanding.
- Stackoverflow is **NOT THE ANSWER**.

Numeric values.

- Integer format is of course “d”. The “i” version is older, and has been deprecated several times.
- Want a decimal point? Then use “f” or maybe one of eEFG. Read the man page.
- And there are various prefixes that you can give these for things like hexadecimal.
- uxXo take an unsigned int and print it as an unsigned int, hexadecimal, or octal.

(more ...)

- There are length modifiers for short (int), long, and long long.
- Then there is width and precision. That is minimum field width.
- As in “%8.4f”. 4 digits of precision and minimum width of 8.
- “%08.4f” pads with leading 0s.
- Width works with strings as well.

(more ...)

- And of course can specify that commas are used for grouping (US standard). See this [page](#).
- That a sign is always printed.
- And if you want your hexadecimal printed as '0x...' then prefix the format with a #.

(more ...)

- And then there is
 - “c” for a single character.
 - “s” expects a ‘\0’ terminated char * array as an argument.
 - “P” prints a pointer (address) like “%#x”.
 - “%%” if you want a percent sign.

Return values

- If a negative value is returned there was an output error. How that can happen without the program being killed I do not know.
- Otherwise the number of characters printed.
- This is really one of the few exceptions to “always check return values.”

wycat (HW4)

- YOU WILL NOT PRINT WITH `printf()`.
- `printf()` is 'slow' by comparison with `fwrite()`.
- DO NOT use `printf()` for the wycat assignment, or any other assignment unless I specify its use.