

# Make

---

---

---

---

---

---

---

---

## make

- The make command automates the process of building files that depend on other files.
- Typically used for program development:
  - runs the compiler only when necessary.
  - uses file access times to decide when it is necessary.
- make can be used for lots of tasks! (not just for programming).

---

---

---

---

---

---

---

---

## Dependency

- File **foo** should be *rebuilt* whenever file **blah** is changed.
  - if **blah** is newer than **foo** should, we need to rebuild **foo**.
- *foo depends on blah*

---

---

---

---

---

---

---

---

## Makefiles

- You have to give make a file containing *rules*.
- Rules include:
  - file dependencies
  - instructions on how to build things (how to create the dependant file).

---

---

---

---

---

---

---

---

## Default input files

if you don't tell make otherwise, it looks for the file named **makefile** for rules.

If there is no file named **makefile**, it looks for the file named **Makefile**.

You can also use: **make -f filename**

---

---

---

---

---


---

---

---

## Rules

This MUST be a tab!!!



```
target : prerequisites  
    command1  
    command2  
    ...
```

target depends on the files listed after the colon.  
commands are unix commands that build a new target file.

---

---

---

---

---

---

---

---

## Simple Example

- This rule would tell make that the file **linecount** depends on the file **foo.c**.
- To build linecount the command “**wc -l foo.c > linecount**” should be run.

```
linecount : foo
    wc -l foo.c > linecount
```

Intro to Unix Spring 2000 Make

7

---

---

---

---

---

---

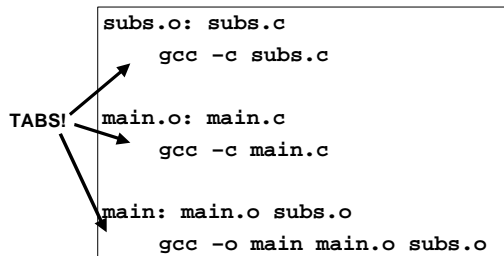
---

---

## Simple Makefile for building a program.

```
subs.o: subs.c
    gcc -c subs.c
main.o: main.c
    gcc -c main.c
main: main.o subs.o
    gcc -o main main.o subs.o
```

TABS!



Intro to Unix Spring 2000 Make

8

---

---

---

---

---

---

---

---

## Make command line (simplified)

```
make [options] [targets]
```

- options include
  - “-f filename” use filename as the Makefile
  - “-n” don’t do anything – just print what needs to be done.
- targets are what should be built (if needed).

Intro to Unix Spring 2000 Make

9

---

---

---

---

---

---

---

---

## Macros

- You can create macros (fancy variables)

```
OBJS = subs.o main.o
```

```
main: $(OBJS)
    gcc -o main $(OBJS)
```

Intro to Unix Spring 2000 Make

10

---

---

---

---

---

---

---

---

## Fancy Stuff

- Macro substitution – evaluates to the value of a macro after some substitutions:

```
SOURCE = main.c subs.c
OBJS = ${SOURCE:.c=.o}
```

now **OBJS** is **main.o subs.o**

Intro to Unix Spring 2000 Make

11

---

---

---

---

---

---

---

---

## Suffix Rules

- General rules for building files that end with some suffix from files with the same name but a different suffix.
- For example, how to build a **.o** file from a **.c** file.
- Special predefined macros help with suffix rules:

**\$<** means current prereq.

**\$@** is the current target.

Intro to Unix Spring 2000 Make

12

---

---

---

---

---

---

---

---

## Example Suffix Rule

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

This rule tells Make how to create a `.o` file any time it has a `.c` file (and needs the `.o` file to build a target).

Intro to Unix Spring 2000 Make

13

---

---

---

---

---

---

---

---

## A Complete Example

```
.c.o:  
gcc -c $<  
  
SOURCE = main.c subs.c  
OBJS = ${SOURCE:.c=.o}  
  
main: $(OBJS)  
gcc -o main $(OBJS)
```

Intro to Unix Spring 2000 Make

14

---

---

---

---

---

---

---

---