CS 6015 Software Engineering

Master of Software Development

SCHOOL OF COMPUTING | THE UNIVERSITY OF UTAH

Spring 2022

Part 1: Syllabus

Software Engineering

- code structure
- testing
  - test coverage
  - continuous integration
- assertions
- debugging techniques
- optimization techniques
- documentation
- design patterns
- revision control

Some Programming Languages, Too

Building **language interpreters** is a rich source of

- example problems for good engineering practice
- insight into programming generally

Programming-language constructs reflect good programming ideas

Syllabus

Text

Code Complete (second edition)
Steve McConnell

Language and environments

C++
command line and **`make`**
Xcode or CLion
Git and GitHub

Syllabus

Text

Code Complete (second edition)
Steve McConnell

part II: *will assign some*
part III: *at leisure*
part IV: *at leisure*
part V: *will assign some*

Language and environments

- C++
- command line and **`make`**
- Xcode or CLion
- Git and GitHub

Why C++

Exposes abstraction and performance issues

Makes everything harder in a pedagogically useful way

Fits well with your other courses this semester

Course Structure

Learning by doing

Mostly one long project with weekly checkpoints

You'll have to

- write code
- read code
- maintain code
- exchange code
- manage code
- document code

The project: **MSDscript** interpreter

Course Organization

All material is on Canvas

Watch videos before each class meeting

Meet at 9:30 (also streamed on Zoom)

Have reading/homework completed before class

My availability

Tuesday morning / Thursday morning / by appointment

MEB 3256 **`mflatt@cs.utah.edu`**

See Zoom link on Canvas

About Me

https://racket-lang.org

Part 2: Multi-file C++ Programs

Writing C++ Code

Use programming environments like Xcode and CLion for developing code

Support command-line tools for sharing code

```
$ c++ main.cpp sum.cpp
$ ./sum
```

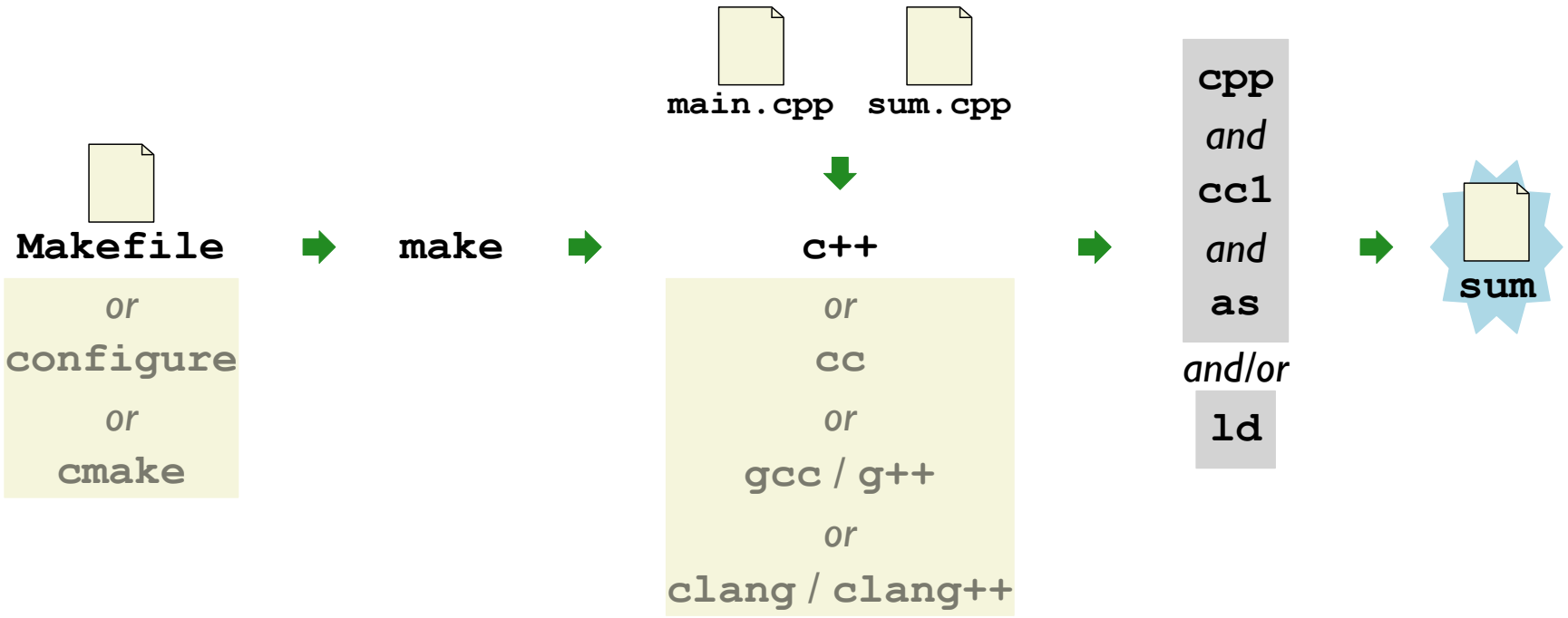
Writing C++ Code

Use programming environments like Xcode and CLion for developing code

Support command-line tools for sharing code

```
$ make
$ ./sum
```

Command-Line Tools



Multi-File C++ Programs

main.cpp

```
int sum(int n);

int array[2] = {1, 2};

int main()  {
  int val = sum(2);
  return val;
}
```

sum.cpp

```
extern int array[];

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

```
$ c++  -o sum  main.cpp sum.cpp
$ ./sum
```

Multi-File C++ Programs

main.cpp

```
int sum(int n);

int array[2] = {1, 2};

int main()  {
  int val = sum(2);
  return val;
}
```

sum.cpp

```
extern int array[];

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

```
$ c++  -o sum  main.cpp sum.cpp
$ ./sum
```

cpp + cc1 + as + ld

Multi-File C++ Programs

main.cpp

```
int sum(int n);

int array[2] = {1, 2};

int main()  {
  int val = sum(2);
  return val;
}
```

sum.cpp

```
extern int array[];

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

```
$ c++ -c  main.cpp
$ c++ -c  sum.cpp
$ c++  -o sum  main.o sum.o
$ ./sum
```

Multi-File C++ Programs

main.cpp

```
int sum(int n);

int array[2] = {1, 2};

int main()  {
  int val = sum(2);
  return val;
}
```

sum.cpp

```
extern int array[];

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

cpp + cc1 + as

```
$ c++ -c  main.cpp
$ c++ -c  sum.cpp
$ c++  -o sum  main.o sum.o
$ ./sum
```

Multi-File C++ Programs

main.cpp

```
int sum(int n);

int array[2] = {1, 2};

int main()  {
  int val = sum(2);
  return val;
}
```

sum.cpp

```
extern int array[];

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

```
$ c++ -c  main.cpp
$ c++ -c  sum.cpp
$ c++  -o sum  main.o sum.o
$ ./sum
```

ld

Why Compile and Link Separately?

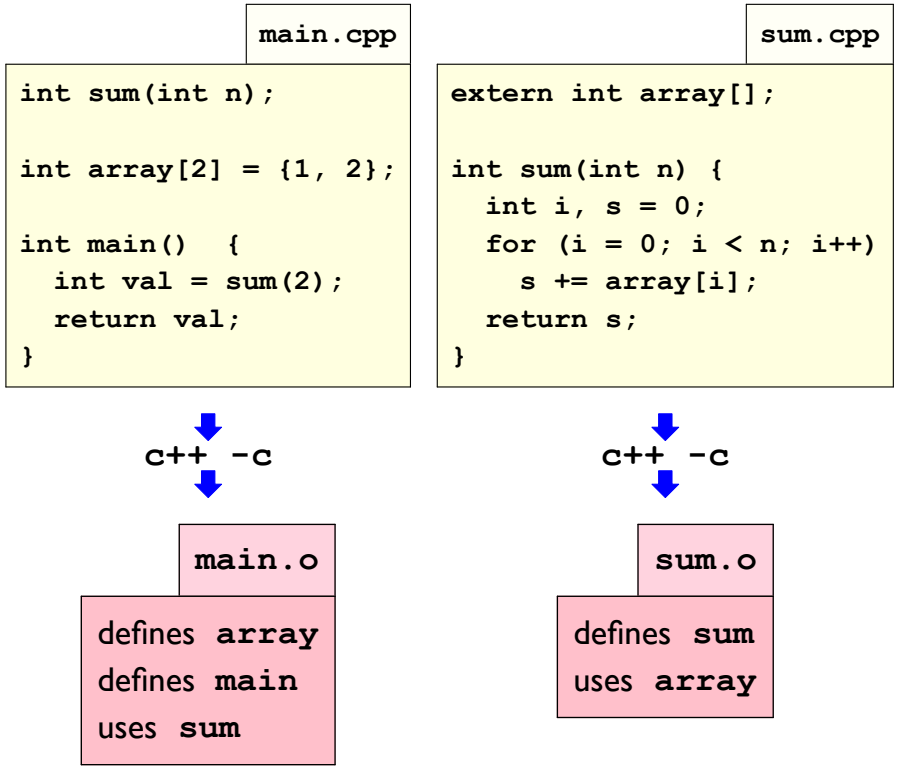
cpp + cc1 + as

```
$ c++ -c  main.cpp
$ c++ -c  sum.cpp
$ c++  -o sum  main.o sum.o
$ ./sum
```

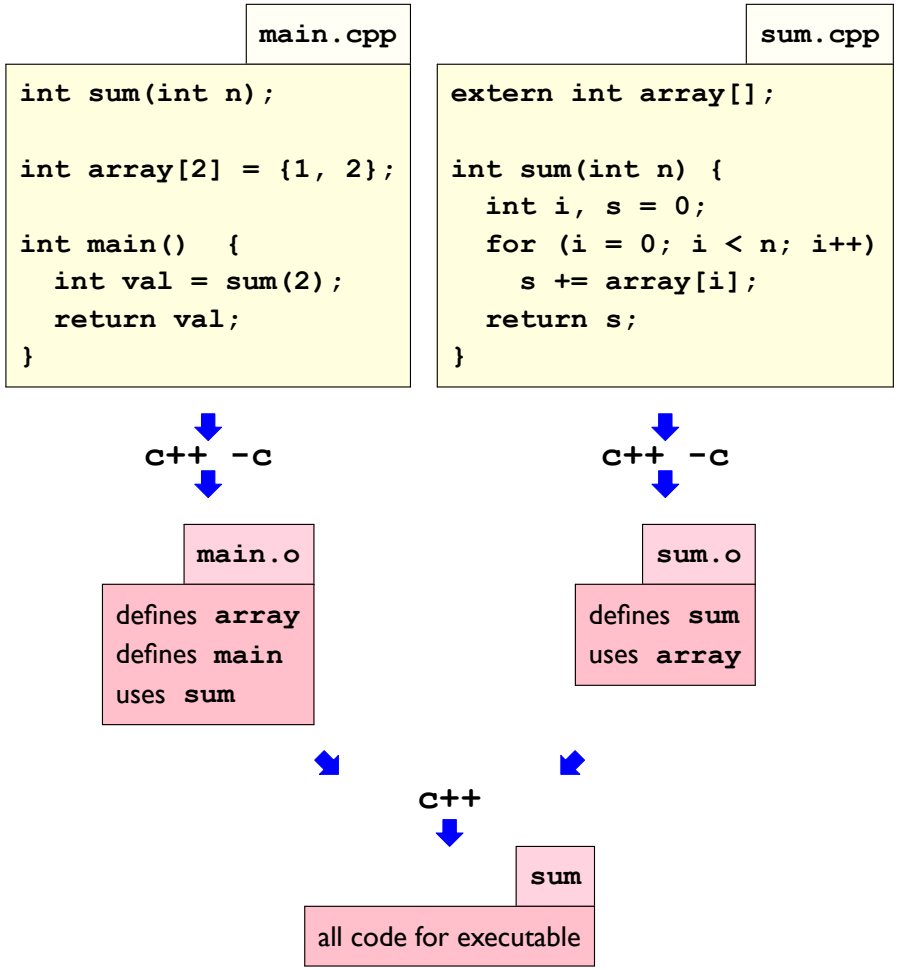
The **cc1** step tends to be the long one

- *Faster:* Don't re-compile if nothing changed
- *Faster:* One **sum.o** with different main programs
- *Smaller:* Sharing **.o** leads to in-memory code sharing

Separate Compilation and Linking



Separate Compilation and Linking



Using Definitions from Other Files

Function:

```
int sum(int n);
```

Data:

```
int array[];
```

Using Definitions from Other Files

Function:

```
extern int sum(int n);
```

Data:

```
extern int array[];
```

`extern` is optional, but especially good practice for data

Providing Definitions to Other Files

Function:

```
int sum(int n) {
  ....
}
```

Data:

```
int array[2] = {1, 2};
```

Providing Definitions to Other Files

Function:

```
int sum(int n) {
  ....
}
```

Data:

```
int array[2];
```

initialization is optional, but especially good practice for data

Declaring and Implementing

It's ok to both declare and implement:

```
extern int a[];
int sum(int n);

....

int a[2] = {1, 2};

int sum(int n) {
    ....
}
```

Declaration and implementation must be consistent

Consistency of Definitions

main.cpp

```
double sum(int n);

double array[2] = {1, 2};

int main()  {
  double val = sum(2) > 0.0;
  return val;
}
```

sum.cpp

```
extern int array[];

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

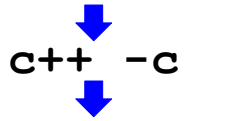
Consistency of Definitions

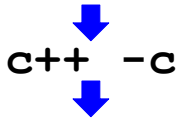
main.cpp

```
double sum(int n);

double array[2] = {1, 2};

int main()  {
  double val = sum(2) > 0.0;
  return val;
}
```

sum.cpp

```
extern int array[];

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```





main.o

defines **array**
defines **main**
uses **sum**

sum.o

defines **sum**
uses **array**

Consistency of Definitions

main.cpp

```
double sum(int n);

double array[2] = {1, 2};

int main()  {
  double val = sum(2) > 0.0;
  return val;
}
```

sum.cpp

```
extern int array[];

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

"Random" result due to **sum** mismatch

Ensuring Consistency

sum.h

```
extern int array[];
int sum(int n);
```

main.cpp

```
#include "sum.h"

int array[2] = {1, 2};

int main()  {
  int val = sum(2);
  return val;
}
```

sum.cpp

```
#include "sum.h"

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

Header files (**.h**) are for *declarations* not *implementations*

Ensuring Consistency

main.cpp

```
#include "sum.h"

int array[2] = {1, 2};

int main()  {
  int val = sum(2);
  return val;
}
```

sum.cpp

```
#include "sum.h"

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

main.cpp

```
extern int array[];
int sum(int n);

int array[2] = {1, 2};

int main()  {
  int val = sum(2);
  return val;
}
```

sum.cpp

```
extern int array[];
int sum(int n);

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

Ensuring Consistency

main.cpp

```
#include "sum.h"

int array[2] = {1, 2};

int main()  {
  int val = sum(2);
  return val;
}
```

sum.cpp

```
#include "sum.h"

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

Things that start with # are expanded by cpp

main.cpp

```
extern int array[];
int sum(int n);

int array[2] = {1, 2};

int main()  {
  int val = sum(2);
  return val;
}
```

sum.cpp

```
extern int array[];
int sum(int n);

int sum(int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
    s += array[i];
  return s;
}
```

Part 3: Makefiles

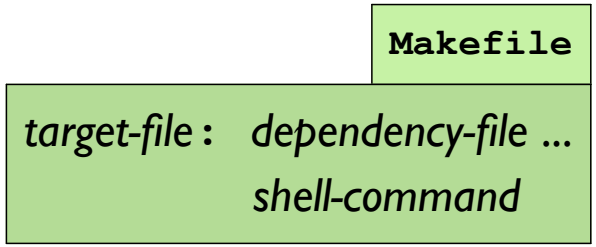
Building with `make`

The **`make`** program is often used to drive C/C++ compilers

- Rebuilds based on file timestamps
- Includes predefined rules for common steps

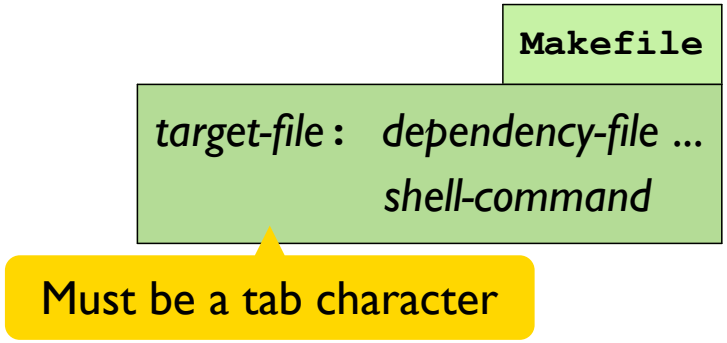
Makefiles

By default, **`make`** reads a file named **`Makefile`**



Makefiles

By default, **`make`** reads a file named **`Makefile`**



Makefiles

By default, **`make`** reads a file named **`Makefile`**

`Makefile`

```
sum: main.cpp sum.cpp
	c++ --std=c++14 -O2 -o sum main.cpp sum.cpp
```

Makefiles

By default, **`make`** reads a file named **`Makefile`**

`Makefile`

```
sum: main.cpp sum.cpp sum.h
	c++ --std=c++14 -O2 -o sum main.cpp sum.cpp
```

Makefiles

By default, **`make`** reads a file named **`Makefile`**

List *all* dependencies

`Makefile`

```
sum: main.cpp sum.cpp sum.h
	c++ --std=c++14 -O2 -o sum main.cpp sum.cpp
```

Makefiles

By default, **`make`** reads a file named **`Makefile`**

`Makefile`

```
SRCS = main.cpp sum.cpp

sum: $(SRCS) sum.h
	c++ --std=c++14 -O2 -o sum $(SRCS)
```

Makefiles

By default, **`make`** reads a file named **`Makefile`**

= defines a variable

`Makefile`

```
SRCS = main.cpp sum.cpp

sum: $(SRCS) sum.h
        c++ --std=c++14 -O2 -o sum $(SRCS)
```

Makefiles

By default, **`make`** reads a file named **`Makefile`**

`Makefile`

```
SRCS = main.cpp sum.cpp

sum: $(SRCS) sum.h
        c++ --std=c++14 -O2 -o sum $(SRCS)
```

`$(var)` is replaced by *var*'s value

Makefiles

By default, **`make`** reads a file named **`Makefile`**

`Makefile`

```
INCS = sum.h

SRCS = main.cpp sum.cpp

sum: $(SRCS) $(INCS)
        c++ --std=c++14 -O2 -o sum $(SRCS)
```

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

sum: $(OBJS)
        c++ --std=c++14 -O2 -o sum $(OBJS)

main.o: main.cpp $(INCS)
        c++ --std=c++14 -O2 -c main.cpp

sum.o: sum.cpp $(INCS)
        c++ --std=c++14 -O2 -c sum.cpp
```

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

sum: $(OBJS)
        c++ --std=c++14 -O2 -o sum $(OBJS)

main.o: main.cpp $(INCS)
        c++ --std=c++14 -O2 -c main.cpp

sum.o: sum.cpp $(INCS)
        c++ --std=c++14 -O2 -c sum.cpp
```

Dependency triggers recursive checking

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXX = c++
CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: main.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c main.cpp

sum.o: sum.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c sum.cpp
```

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: main.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c main.cpp

sum.o: expr.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c sum.cpp
```

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: main.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c main.cpp

sum.o: expr.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c sum.cpp
```

CXX is predefined

Makefiles

CXXFLAGS is also predefined, but can be redefined

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: main.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c main.cpp

sum.o: expr.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c sum.cpp
```

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: main.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c $<

sum.o: sum.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c $<
```

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: main.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c $<

sum.o: sum.cpp $(INCS)
        $(CXX) $(CXXFLAGS) -c $<
```

$< means "the first dependency"

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: main.cpp $(INCS)

sum.o: sum.cpp $(INCS)
```

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: main.cpp $(INCS)

sum.o: sum.cpp $(INCS)
```

`$(CXX) $(CXXFLAGS) -c $<` is the default command for getting from `.cpp` to `.o`

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: $(INCS)

sum.o: $(INCS)
```

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: $(INCS)

sum.o: $(INCS)
```

Built-in meta rule:
If want `.o` and have `.cpp`,
then use that default rule

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: $(INCS)

sum.o: $(INCS)
```

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: $(INCS)

sum.o: $(INCS)
```

make by default builds first target

Makefiles

Makefile

```
INCS = sum.h

OBJS = main.o sum.o

CXXFLAGS = --std=c++14 -O2

sum: $(OBJS)
        $(CXX) $(CXXFLAGS) -o sum $(OBJS)

main.o: $(INCS)

sum.o: $(INCS)
```

make main.o
builds this one

Part 4: Git and GitHub

Git and GitHub

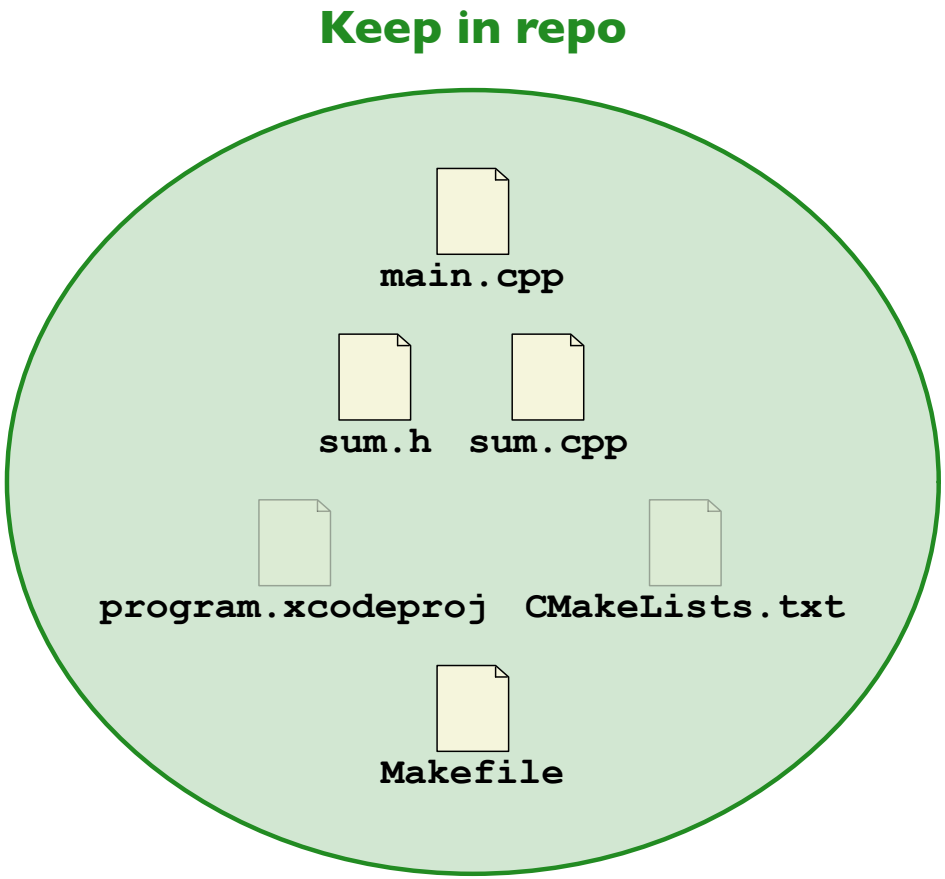
Create a new repo on GitHub for your project

... and consider using "msdscript" in the name

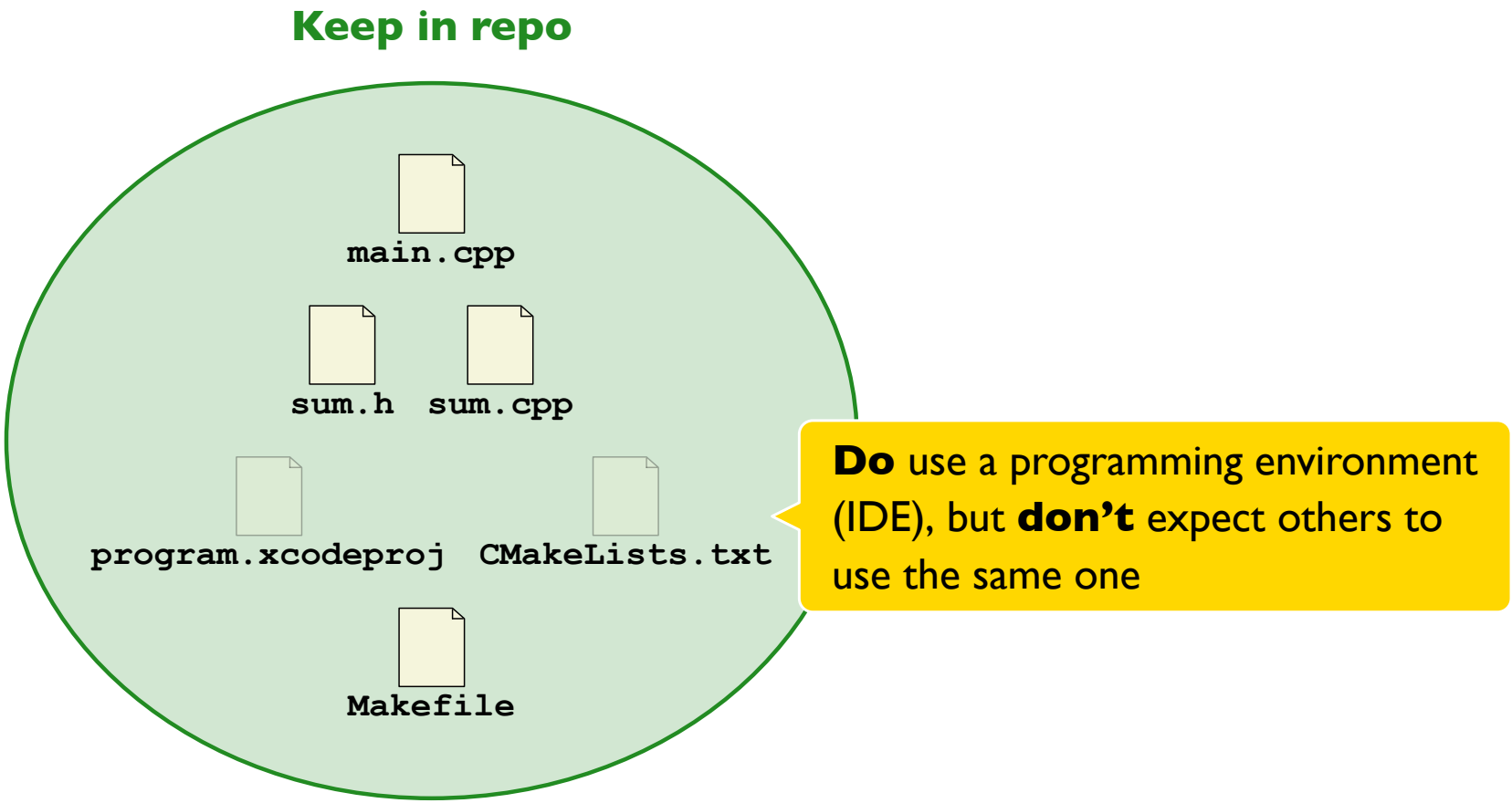
This is **not** your existing **`UtahMSD/`*`fullName`*** repo

Unlike **`UtahMSD/`*`fullName`***, this will will be expected to have a readable commit history with good messages

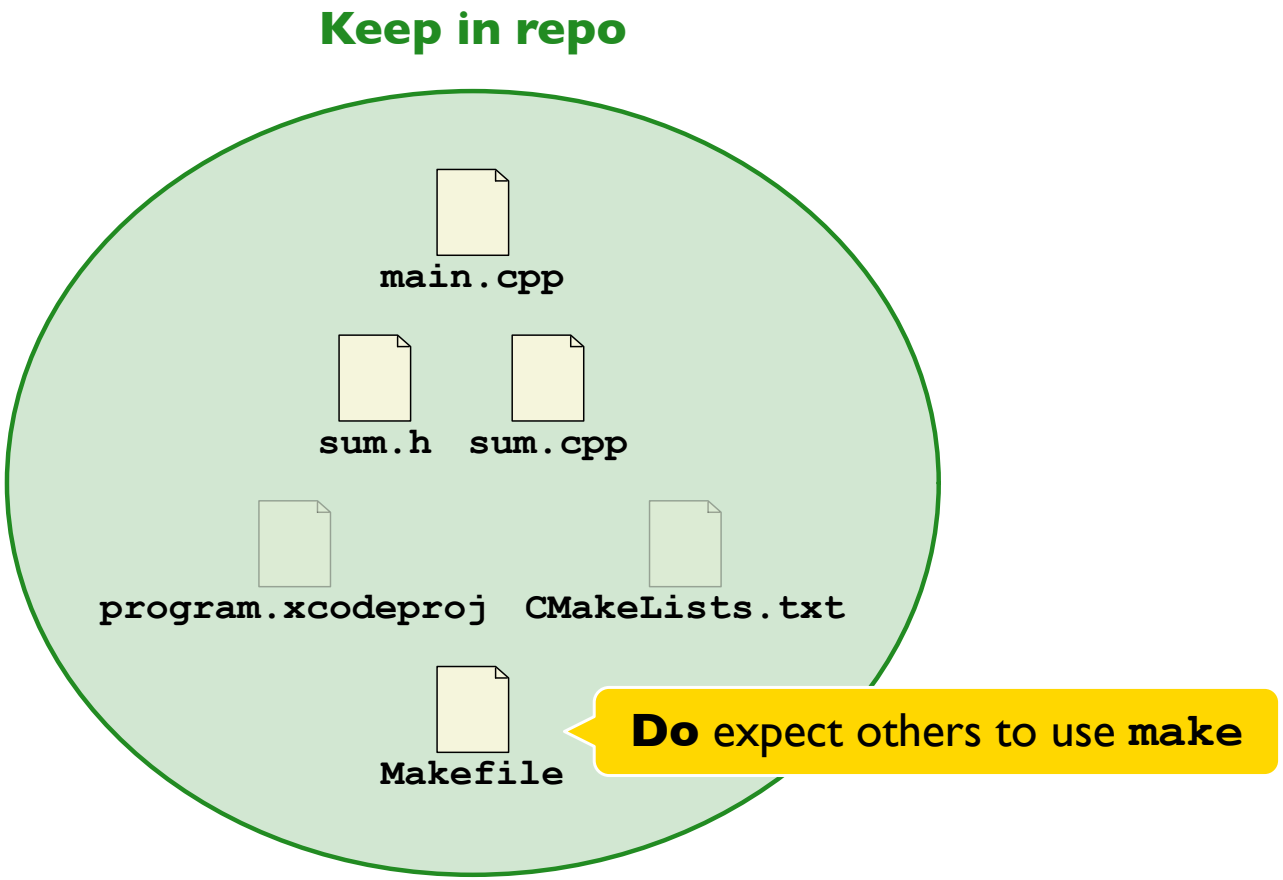
C++ and Files


Keep in repo
main.cpp
sum.h
sum.cpp
program.xcodeproj
CMakeLists.txt
Makefile

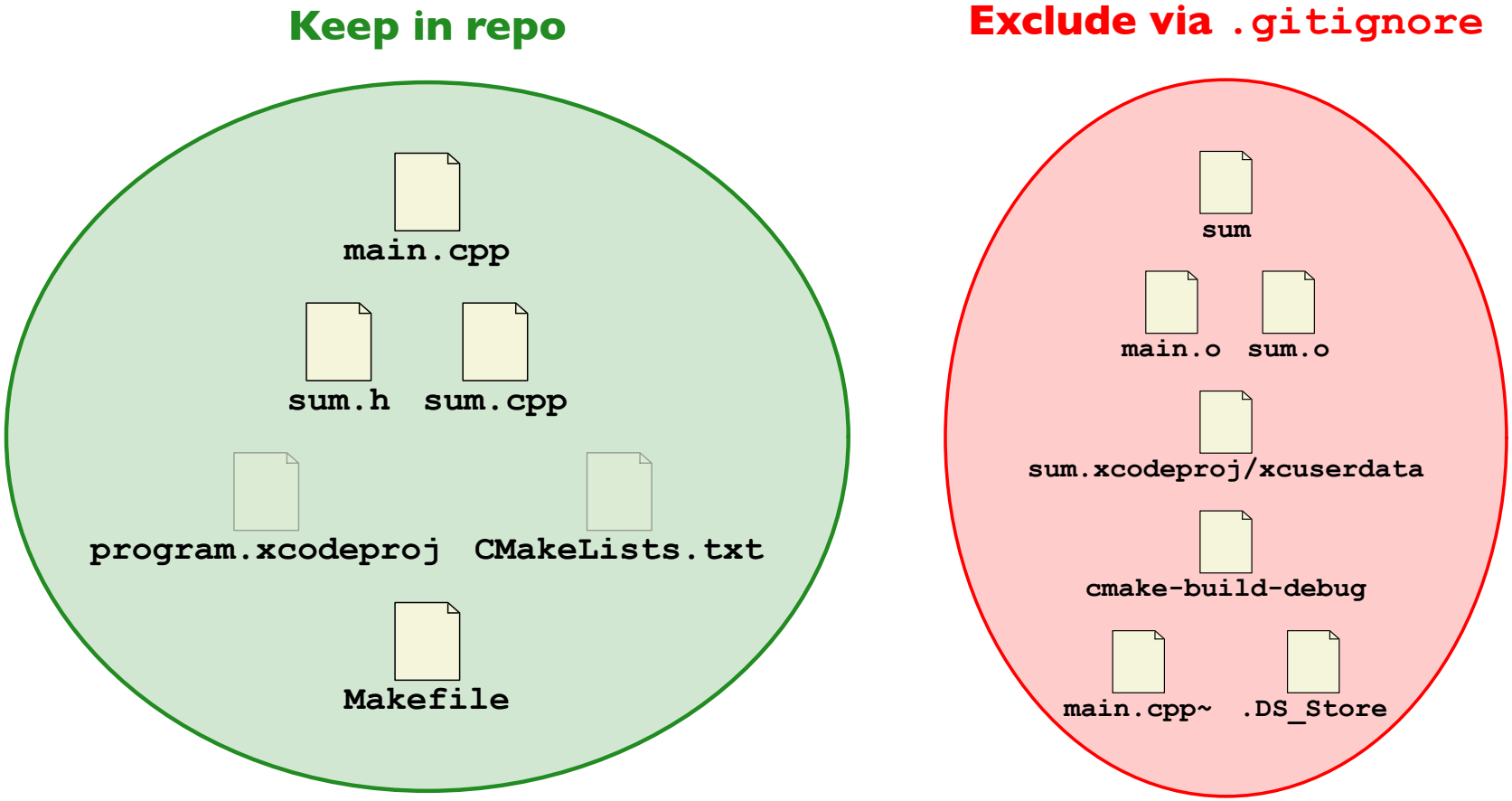
C++ and Files


Keep in repo
main.cpp
sum.h
sum.cpp
program.xcodeproj
CMakeLists.txt
Makefile
Do use a programming environment (IDE), but don't expect others to use the same one

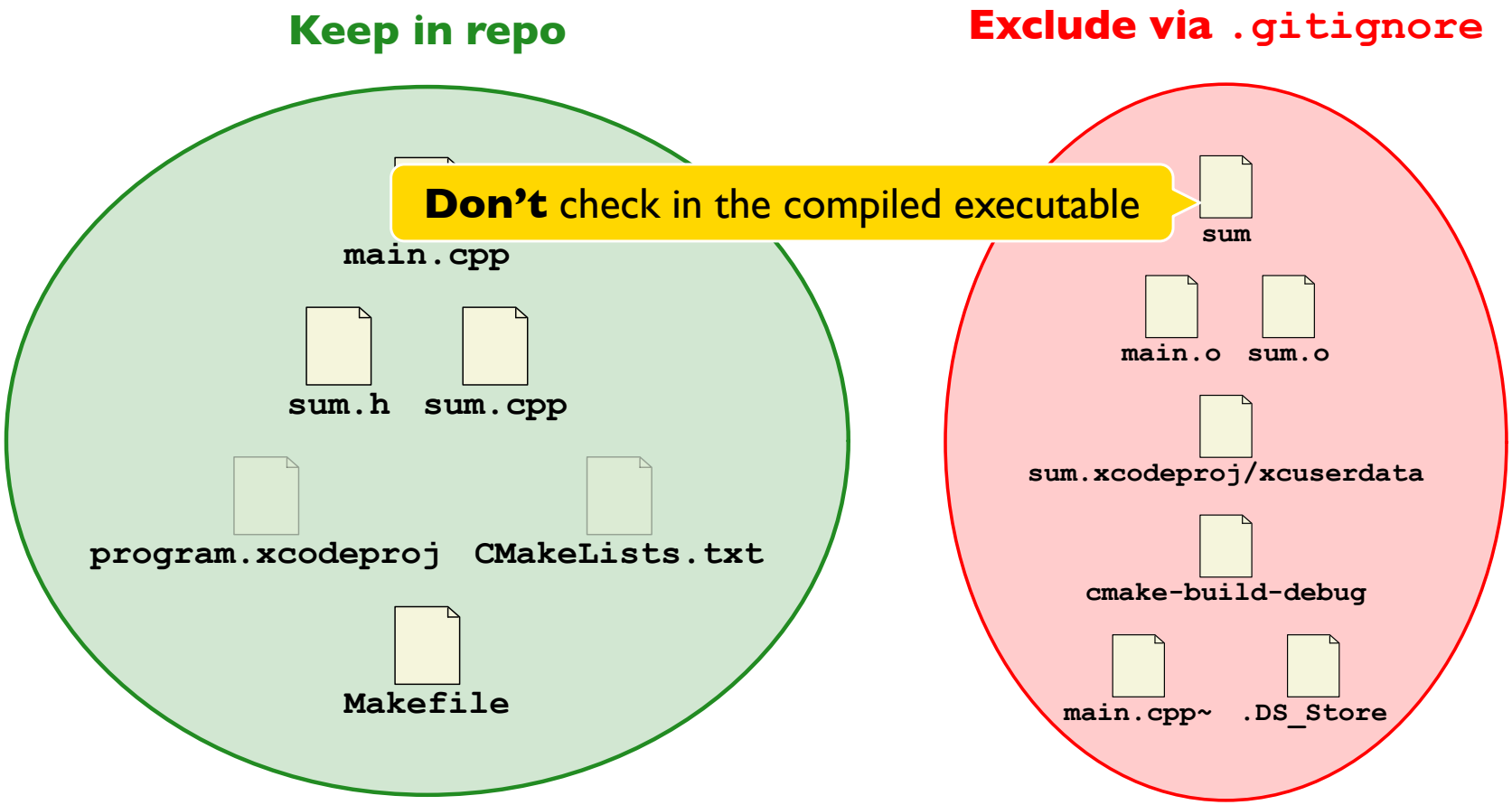
C++ and Files


Keep in repo
main.cpp
sum.h
sum.cpp
program.xcodeproj
CMakeLists.txt
Makefile
Do expect others to use make

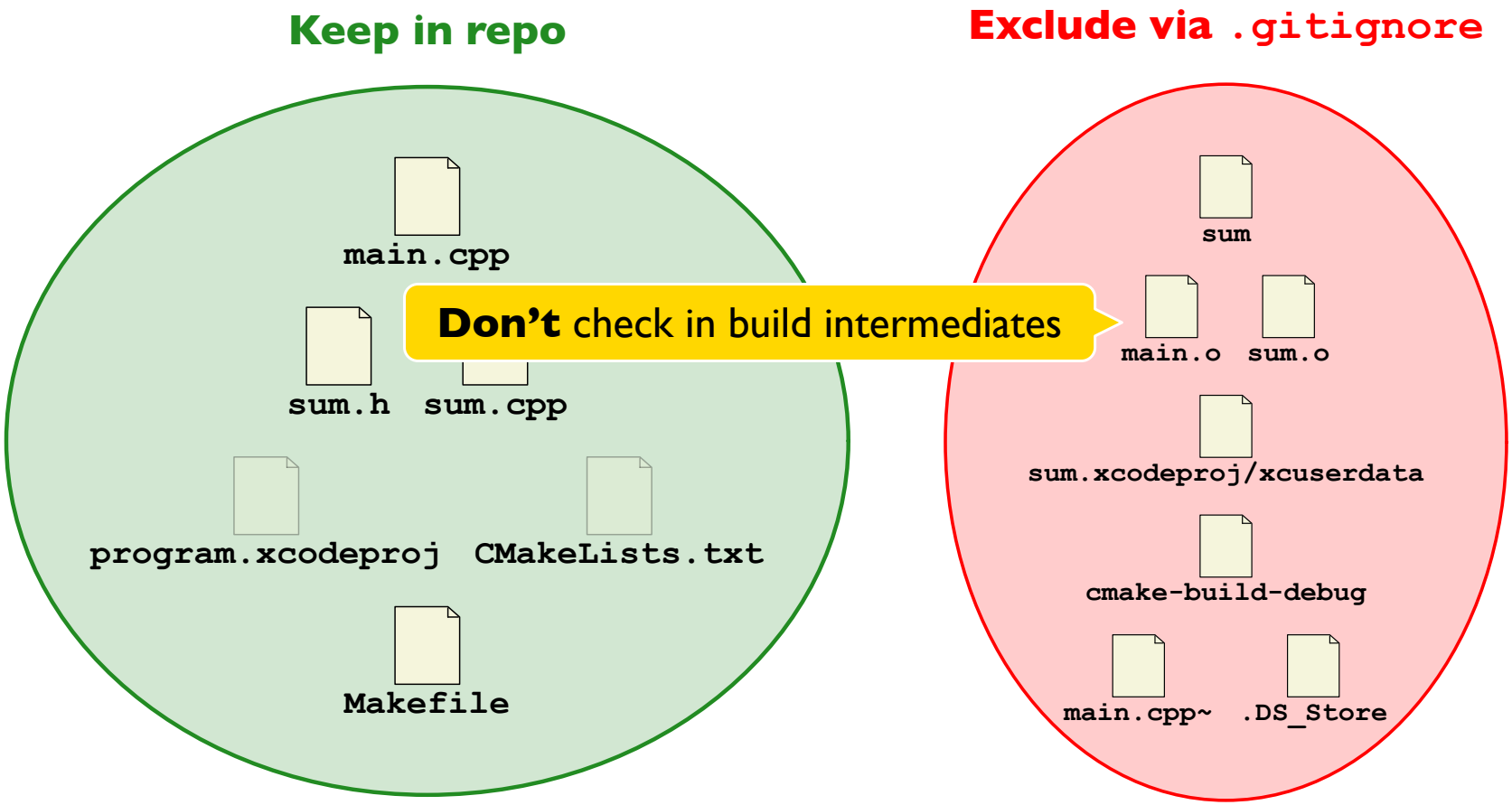
C++ and Files


Keep in repo
main.cpp
sum.h
sum.cpp
program.xcodeproj
CMakeLists.txt
Makefile
Exclude via .gitignore
sum
main.o
sum.o
sum.xcodeproj/xcuserdata
cmake-build-debug
main.cpp~
.DS_Store

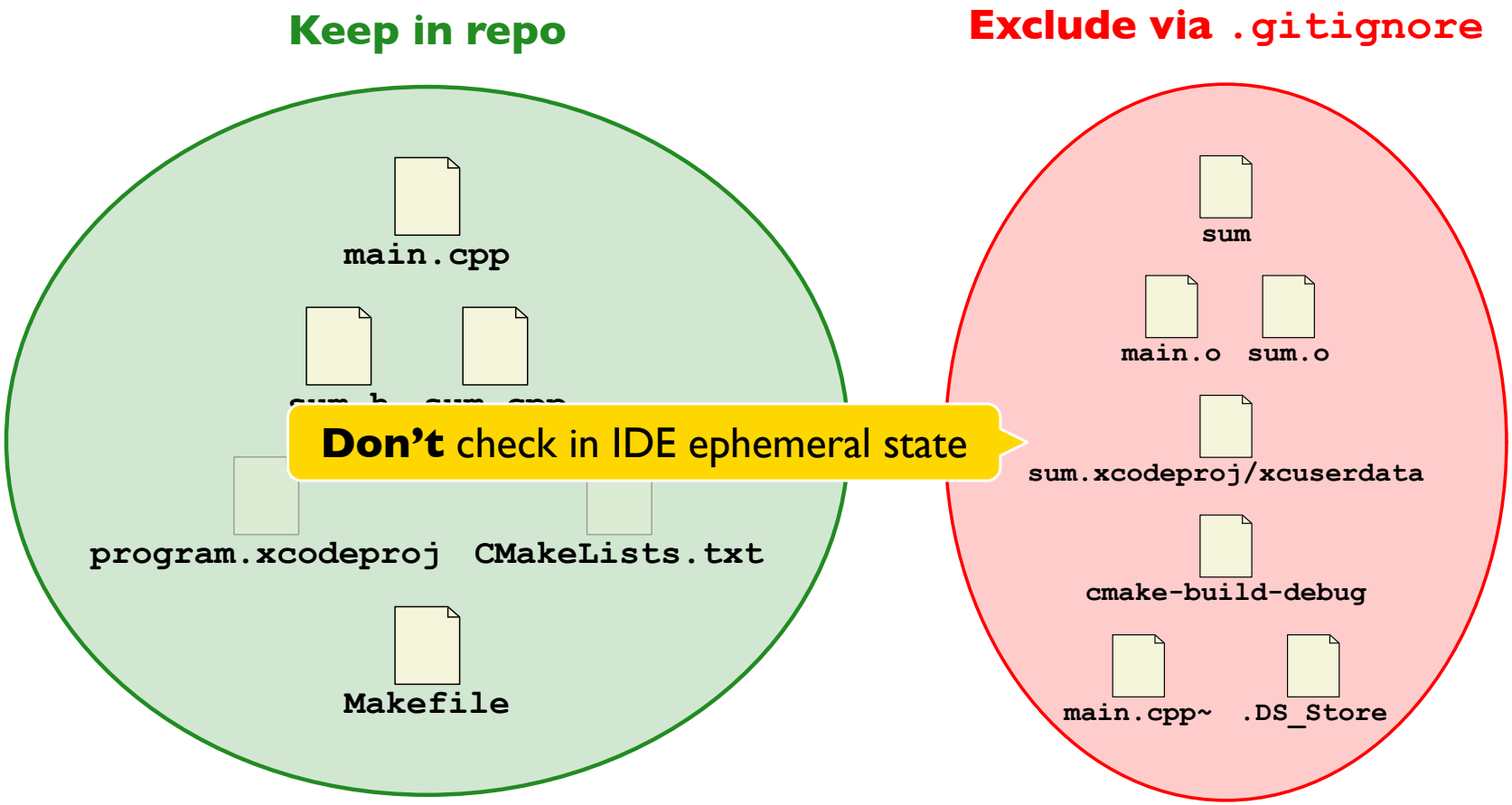
C++ and Files


Keep in repo
Exclude via .gitignore
Don't check in the compiled executable
main.cpp
sum.h
sum.cpp
program.xcodeproj
CMakeLists.txt
Makefile
sum
main.o
sum.o
sum.xcodeproj/xcuserdata
cmake-build-debug
main.cpp~
.DS_Store

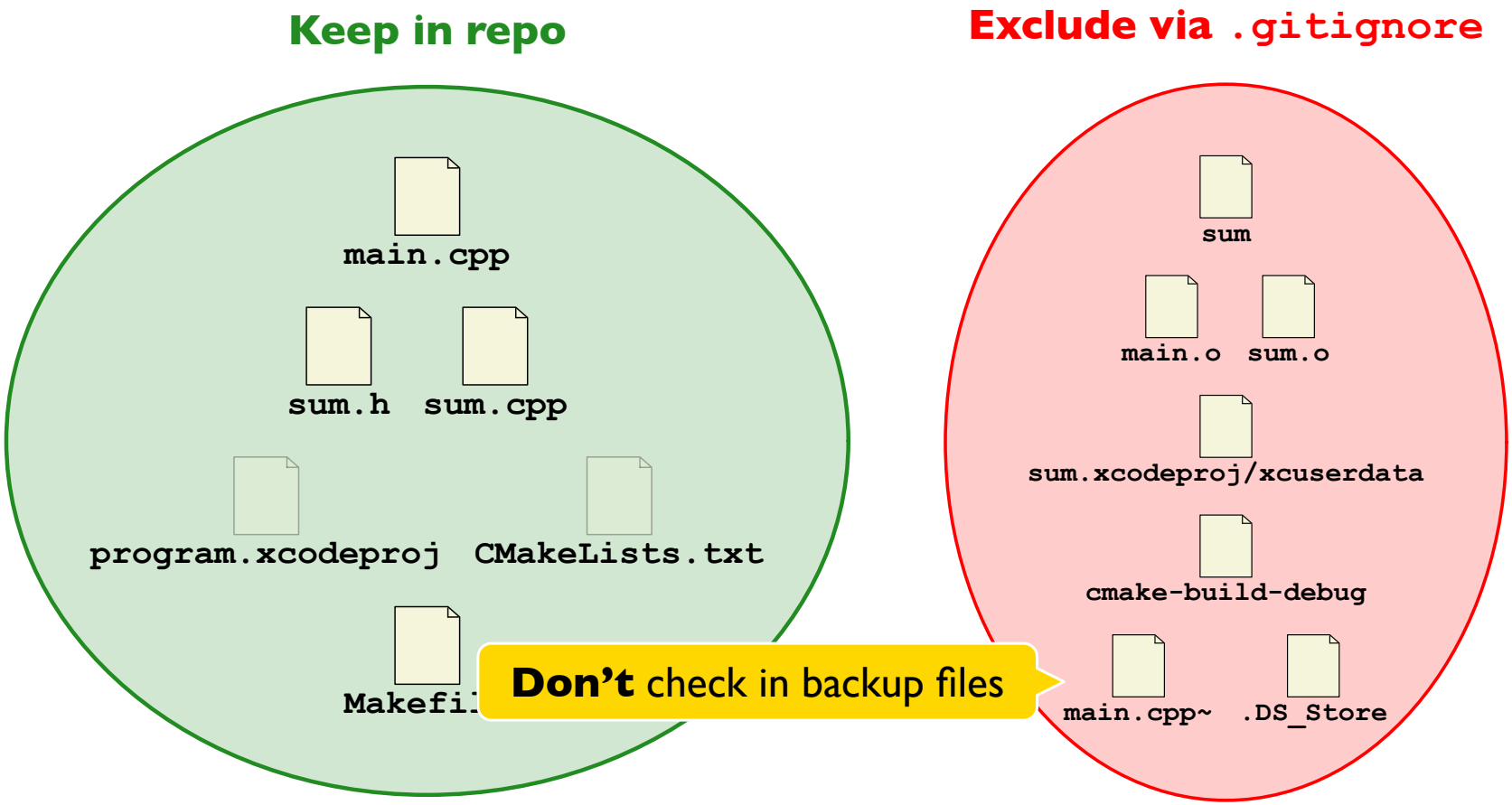
C++ and Files


Keep in repo
Exclude via .gitignore
main.cpp
sum.h
sum.cpp
program.xcodeproj
CMakeLists.txt
Makefile
Don't check in build intermediates
sum
main.o
sum.o
sum.xcodeproj/xcuserdata
cmake-build-debug
main.cpp~
.DS_Store

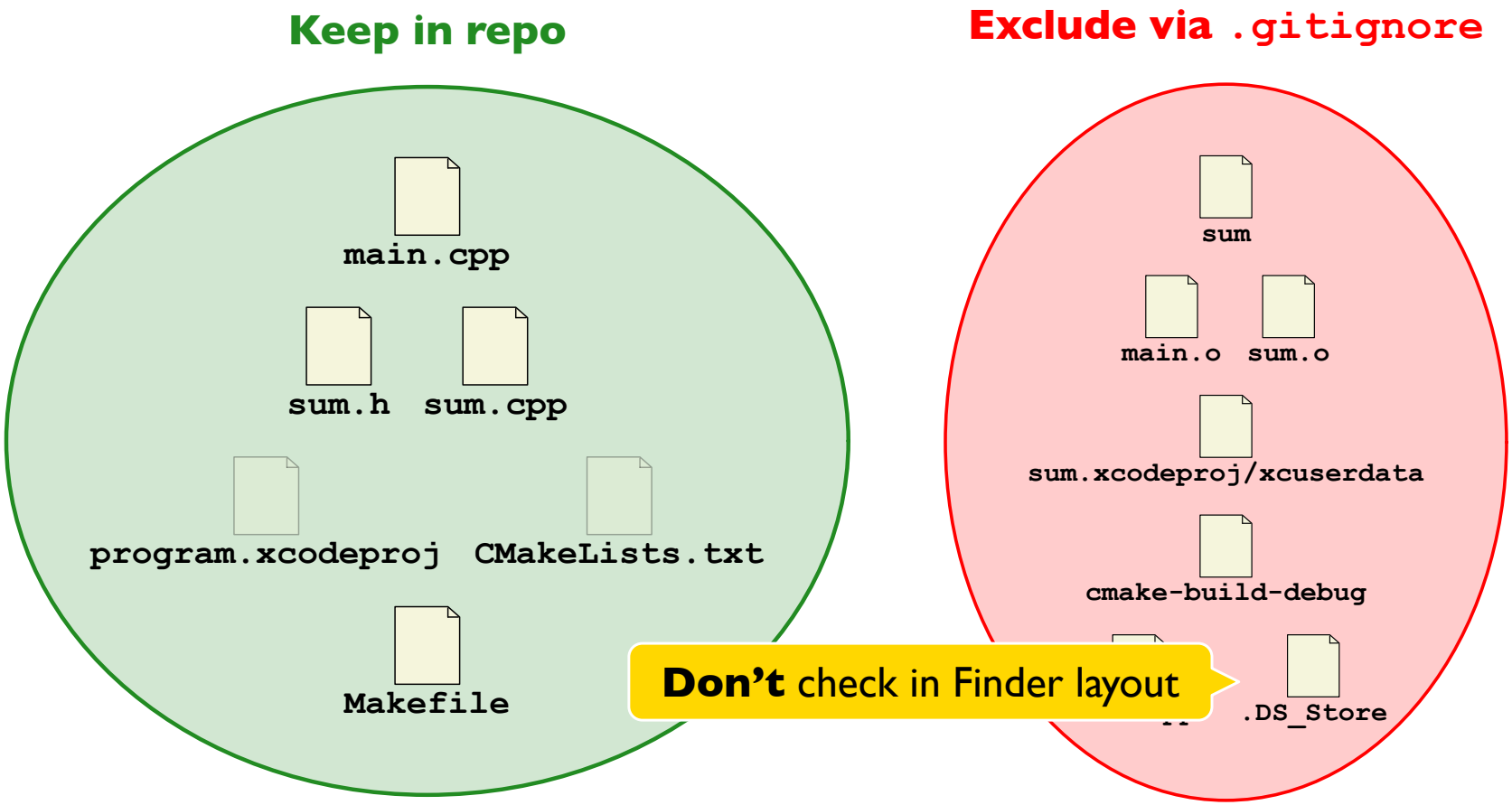
C++ and Files


Keep in repo
Exclude via .gitignore
main.cpp
sum.h
sum.cpp
program.xcodeproj
CMakeLists.txt
Makefile
Don't check in IDE ephemeral state
sum
main.o
sum.o
sum.xcodeproj/xcuserdata
cmake-build-debug
main.cpp~
.DS_Store

C++ and Files


Keep in repo
Exclude via .gitignore
main.cpp
sum.h
sum.cpp
program.xcodeproj
CMakeLists.txt
Makefi
Don't check in backup files
sum
main.o
sum.o
sum.xcodeproj/xcuserdata
cmake-build-debug
main.cpp~
.DS_Store

C++ and Files


Keep in repo
Exclude via .gitignore
main.cpp
sum.h
sum.cpp
program.xcodeproj
CMakeLists.txt
Makefile
sum
main.o
sum.o
sum.xcodeproj/xcuserdata
cmake-build-debug
.DS_Store
Don't check in Finder layout

Creating a Git Repository

- At **`github.com`**, click the **+** in the top right and select **New repository**

  Pick your account as the owner — *not* **UtahMSD**

  Since it's for homework, make the repository **Private**

- On your machine, make a directory for your repository, **`cd`** there, put files there incuding **`.gitignore`**, and use

```
$ git init
$ git add .
$ git commit -m "initial version"
$ git branch -M main
$ git remote add origin git@github.com:user/repo
$ git push -u origin main
```

Creating a Git Repository

- At **`github.com`**, click the **+** in the top right and select **New repository**

  Pick your account as the owner — *not* **UtahMSD**

  Since it's for homework, make the repository **Private**

- On your machine, make a directory for your repository, **`cd`** there, put files there incuding **`.gitignore`**, and use

```
$ git init
$ git add .
$ git commit -m "initial version"
$ git branch -M main
$ git remote add origin git@github.com:user/repo
$ git push -u origin main
```

Creates opaque **`.git`** subdirectory

Creating a Git Repository

- At **`github.com`**, click the **+** in the top right and select **New repository**

  Pick your account as the owner — *not* **UtahMSD**

  Since it's for homework, make the repository **Private**

- On your machine, make a directory for your repository, **`cd`** there, put files there incuding **`.gitignore`**, and use

```
$ git init
$ git add .
$ git commit -m "initial version"
$ git branch -M main
$ git remote add origin git@github.com:user/repo
$ git push -u origin main
```

Stages files to be included

Creating a Git Repository

- At **`github.com`**, click the **+** in the top right and select **New repository**

  Pick your account as the owner — *not* **UtahMSD**

  Since it's for homework, make the repository **Private**

- On your machine, make a directory for your repository, **`cd`** there, put files there incuding **`.gitignore`**, and use

```
$ git init
$ git add .
$ git commit -m "initial version"
$ git branch -M main
$ git remote add origin git@github.com:user/repo
$ git push -u origin main
```

Actually adds files to the repo

Creating a Git Repository

- At **`github.com`**, click the **+** in the top right and select **New repository**

  Pick your account as the owner — *not* **UtahMSD**

  Since it's for homework, make the repository **Private**

- On your machine, make a directory for your repository, **`cd`** there, put files there incuding **`.gitignore`**, and use

```
$ git init
$ git add .
$ git commit -m "initial version"
$ git branch -M main
$ git remote add origin git@github.com:user/repo
$ git push -u origin main
```

Renames default branch to **`main`**

Creating a Git Repository

- At **`github.com`**, click the **+** in the top right and select **New repository**

  Pick your account as the owner — *not* **UtahMSD**

  Since it's for homework, make the repository **Private**

- On your machine, make a directory for your repository, **`cd`** there, put files there incuding **`.gitignore`**, and use

```
$ git init
$ git add .
$ git commit -m "initial version"
$ git branch -M main
$ git remote add origin git@github.com:user/repo
$ git push -u or
```

Points repo to GitHub as "origin"

Creating a Git Repository

- At **`github.com`**, click the **+** in the top right and select **New repository**

  Pick your account as the owner — *not* **UtahMSD**

  Since it's for homework, make the repository **Private**

- On your machine, make a directory for your repository, **`cd`** there, put files there incuding **`.gitignore`**, and use

```
$ git init
$ git add .
$ git commit -m "initial version"
$ git branch -M main
$ git remote add origin git@github.com:user/repo
$ git push -u origin main
```

Uploads to sync GitHub copy

Creating a Git Repository

- At **`github.com`**, click the **+** in the top right and select **New repository**

  Pick your account as the owner — *not* **UtahMSD**

  Since it's for homework, make the repository **Private**

- On your machine, make a directory for your repository, **`cd`** there, put files there incuding **`.gitignore`**, and use

```
$ git init
$ git add .
$ git commit -m "initial version"
$ git branch -M main
$ git remote add origin git@github.com:user/repo
$ git push -u origin main
```

Using Git to Share Code

The **`main`** branch of your Git repository is for communicating code

- The **`main`** branch is not a backup mechanism

  So, **don't** have a history in **`main`** that looks like this:

```
commit a045f9 first cut
commit b788cd part way there
commit 9345ab I was confused
commit cd7723 most tests now pass
...
```

  A commit on the **`main`** branch should generally be a working version

- But you should *absolutely* back up your work along the way, and a branch other than **`main`** is a fine way to backup work

A Git Workflow

- Create a **`work`** branch:

```
$ git branch -d work
$ git branch work
$ git checkout work
```

A Git Workflow

- Create a **`work`** branch:

```
$ git branch -d work
$ git branch work
$ git checkout work
```

Deletes existing **`work`** branch

A Git Workflow

- Create a **`work`** branch:

```
$ git branch -d work
$ git branch work
$ git checkout work
```

"Copies" current branch to **`work`**

A Git Workflow

- Create a **`work`** branch:

```
$ git branch -d work
$ git branch work
$ git checkout work
```

Switches to **`work`** branch

A Git Workflow

- Create a **`work`** branch:

```
$ git branch -d work
$ git branch work
$ git checkout work
```

- Make changes and periodically save work

```
$ git add .
$ git commit -m "whatever... work in progress"
$ git push origin work
```

- At a working state, switch back to **`main`**

```
$ git checkout main
$ git merge --squash work
$ git commit -m "nice description for others to read"
```

A Git Workflow

- Create a **`work`** branch:

```
$ git branch -d work
$ git branch work
$ git checkout work
```

- Make changes and periodically save work

```
$ git add .
$ git commit -m "whatever... work in progress"
$ git push origin work
```

- At a working state, switch back to **`main`**

```
$ git checkout main
$ git merge --squash work
$ git commit -m "nice description for others to read"
```

Switches back to **`main`** — changes files!

A Git Workflow

- Create a **`work`** branch:

```
$ git branch -d work
$ git branch work
$ git checkout work
```

- Make changes and periodically save work

```
$ git add .
$ git commit -m "whatever... work in progress"
$ git push origin work
```

- At a working state, switch back to **`main`**

```
$ git checkout main
$ git merge --squash work
$ git commit -m "nice description for others to read"
```

Stages all **`work`** changes in **`main`**

A Git Workflow

- Create a **`work`** branch:

```
$ git branch -d work
$ git branch work
$ git checkout work
```

- Make changes and periodically save work

```
$ git add .
$ git commit -m "whatever... work in progress"
$ git push origin work
```

- At a working state, switch back to **`main`**

```
$ git checkout main
$ git merge --squash work
$ git commit -m "nice description for others to read"
```

Part 5: C++ Miscellaneous

C++ Miscellaneous

- Use the type `std::string` for strings

```
#include <string>

std::string greeting = "hello";
```

- `char*` can be cast/auto-converted to `std::string`
- Use == on `std::string` values, but not `char*` values

C++ Miscellaneous

- Use `std::cout` for output

```
#include <iostream>

std::cout << "hello world\n"
```

Don't forget `"\n"`!

C++ Miscellaneous

- Use `std::cerr` for error-message output

```
#include <iostream>

std::cerr << "wrong\n"
```

Distinguishing regular output from error output turns out to make things work better