15-440 Recitation 1: SVN and Makefiles

Vijay Vasudevan
CMU Computer Science
Fall 2009

Announcements

- Project 1 Stage 0 due **tomorrow (before class)**
  - Commit a Makefile and source code that builds a server and client binary
  - SVN + Makefiles today
- Rest of Project 1 due September 17th
  - Start early!
  - Ask questions early!

Recitation Mechanics

- 1) These are *your* recitations.
  - We've got a schedule. It's flexible.
  - Ask questions, make comments, …
  - 1 part lecture, 1 part "public office hours" (homework questions? Sure! Project questions? Great!)
- 2) These aren't the final answers
  - Recitations culled from our experience, other faculty, friends in industry, books, etc.
  - We're always looking for better ideas/tools/practices/etc. If you have some, please share.

Recitation Overview

- Today: Intro & Revision Control
  - Managing your source code wisely
- Makefiles and automation 1
  - Automate the boring stuff!
- Design: Modularity and Testability
  - Managing 1000 LoC != 100 LoC
- Debugging: Techniques & Tools
- Automation 2: Scripting

Resources

- Some great books:
  - The Pragmatic Programmer
  - The Practice of Programming
  - Writing Solid Code
- Recitation notes:
  - http://www.cs.cmu.edu/~dga/systems-se.pdf
  - Please don't redistribute: They're very preliminary!

Outline

- Resources
- **Revision Control**
- Makefiles
- General Q&A

Revision Control

- Before you write a line of code…
- Use subversion/CVS/git/etc.
- Provides access to all old versions of your code
  - No more “cp file.cpp file.cpp. 2009-09-17-129pm-oh-god-please-let-this-work”

What is revision control?

- A repository that stores each version
- You explicitly “check out” and “check in” code and changes.
- ```
  unix35:~/tmp> svn checkout https://
  moo.cmcl.cs.cmu.edu/svn/systems-se
  A    systems-se/related.tex
  A    systems-se/acks.tex
  A    systems-se/tinylang.tex
  A    systems-se/emacs.tex
  ….
  ```
  ```

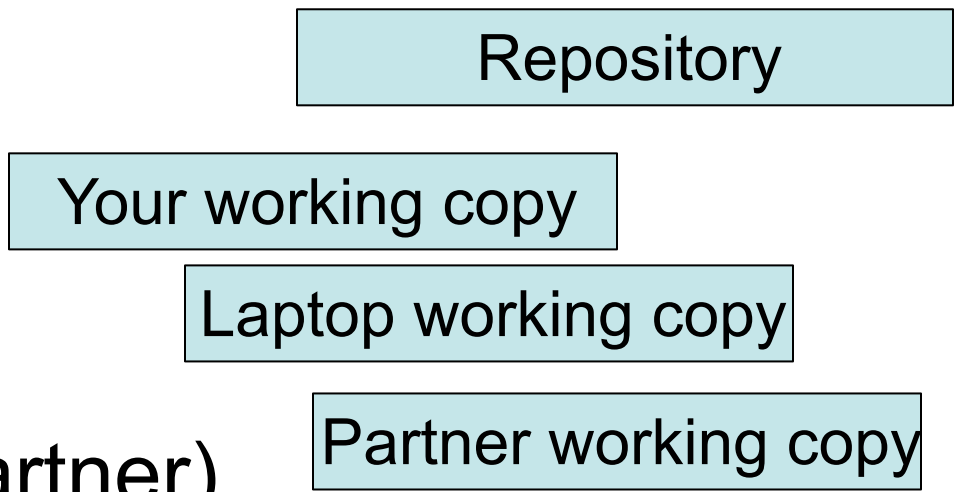
Why do I want it?

- Super-undo: Go to arbitrary versions
  - rm –rf your source tree? No problem!
- Tracking changes /“why did this break?”
- Concurrent development
- Snapshots
  - Turning in the assignment: just make a snapshot when you want, and we’ll grade that. You can keep developing afterwards.
  - Useful, e.g., for optimization contest, or for making sure you have *something* working.

“You’ve sold me. What should I know about it?”

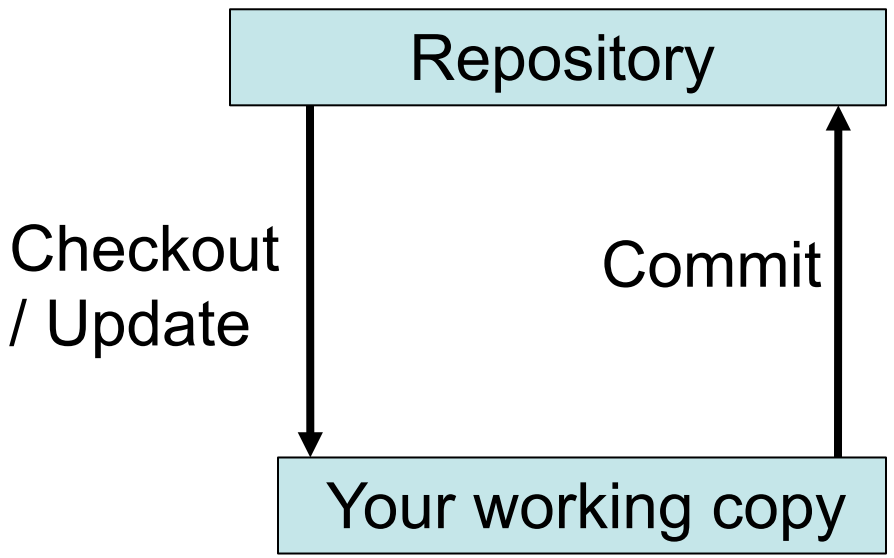
The repository

- Master copy of code is separate from what you work on
- You can have multiple working copies checked out. (So can your partner)

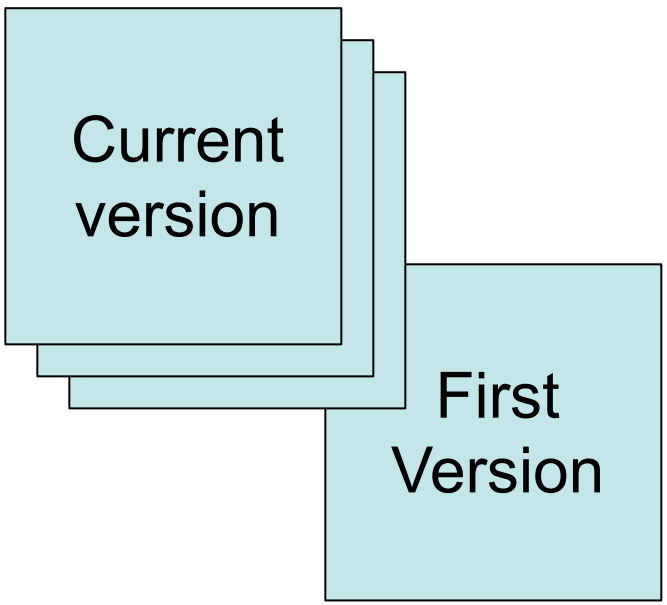


Check out and commit

- Explicitly synchronize with the repository



Every revision is available



And you can see what changed

Revision control lets you note (and then see) what you changed:

> svn log gtcd.cc

```
r986 | ntolia | 2006-08-01 17:13:38 -0400 (Tue, 01 Aug 2006) | 6 lines


This allows the sp to get rid of chunks early before a transfer is
   complete.
Useful when a file is requested in-order and the file size > mem cache
   size
```

And makes it easy to go back to other versions:

```
------------------------------------------------------------------------
r987 | ntolia | 2006-08-02 13:16:21 -0400 (Wed, 02 Aug 2006) | 1 line

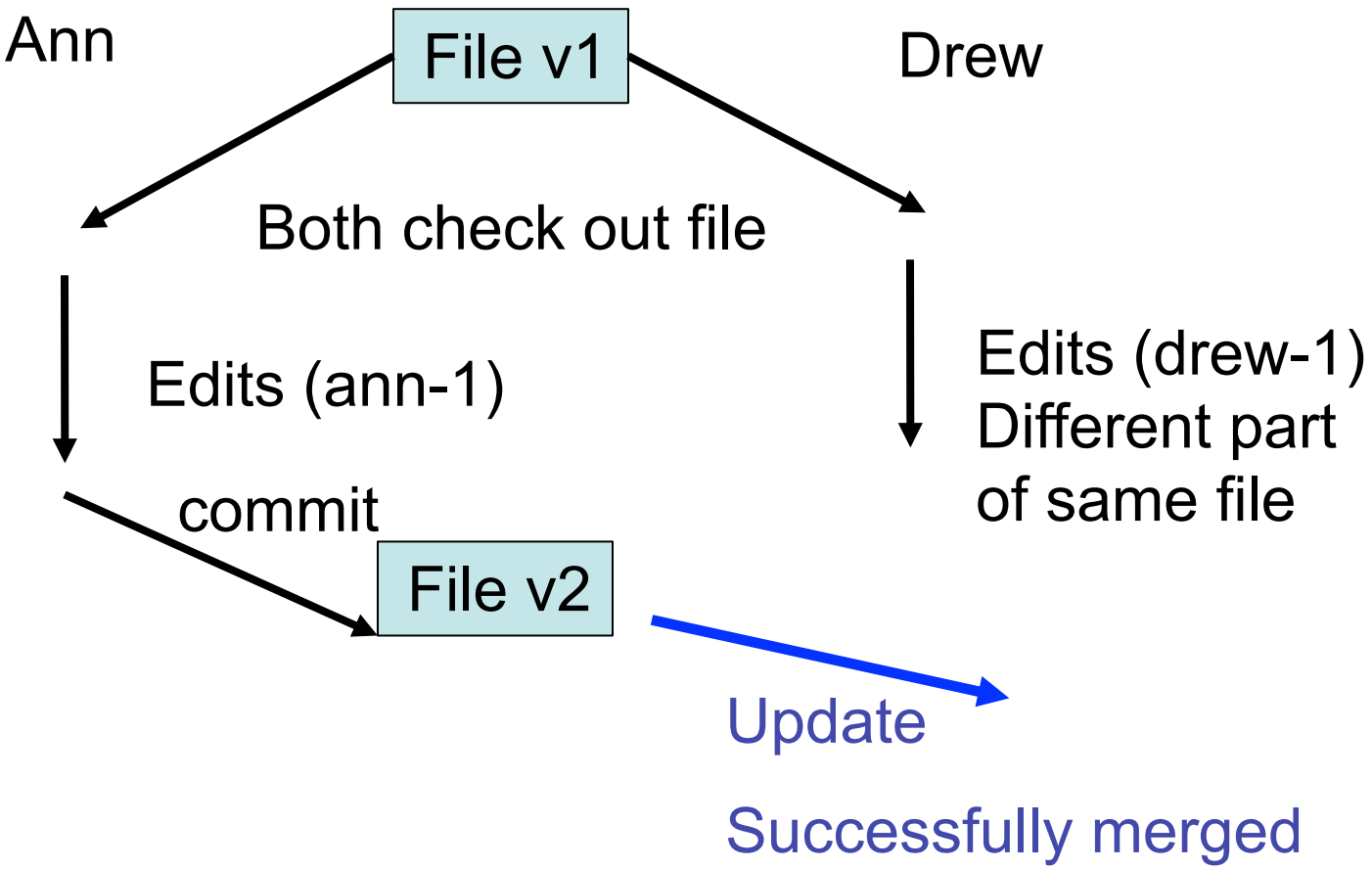

After much thought, I am reverting the last patch. We will need to
   revisit the
issue when we think about DOT on storage-limited clients
```

Concurrent Development

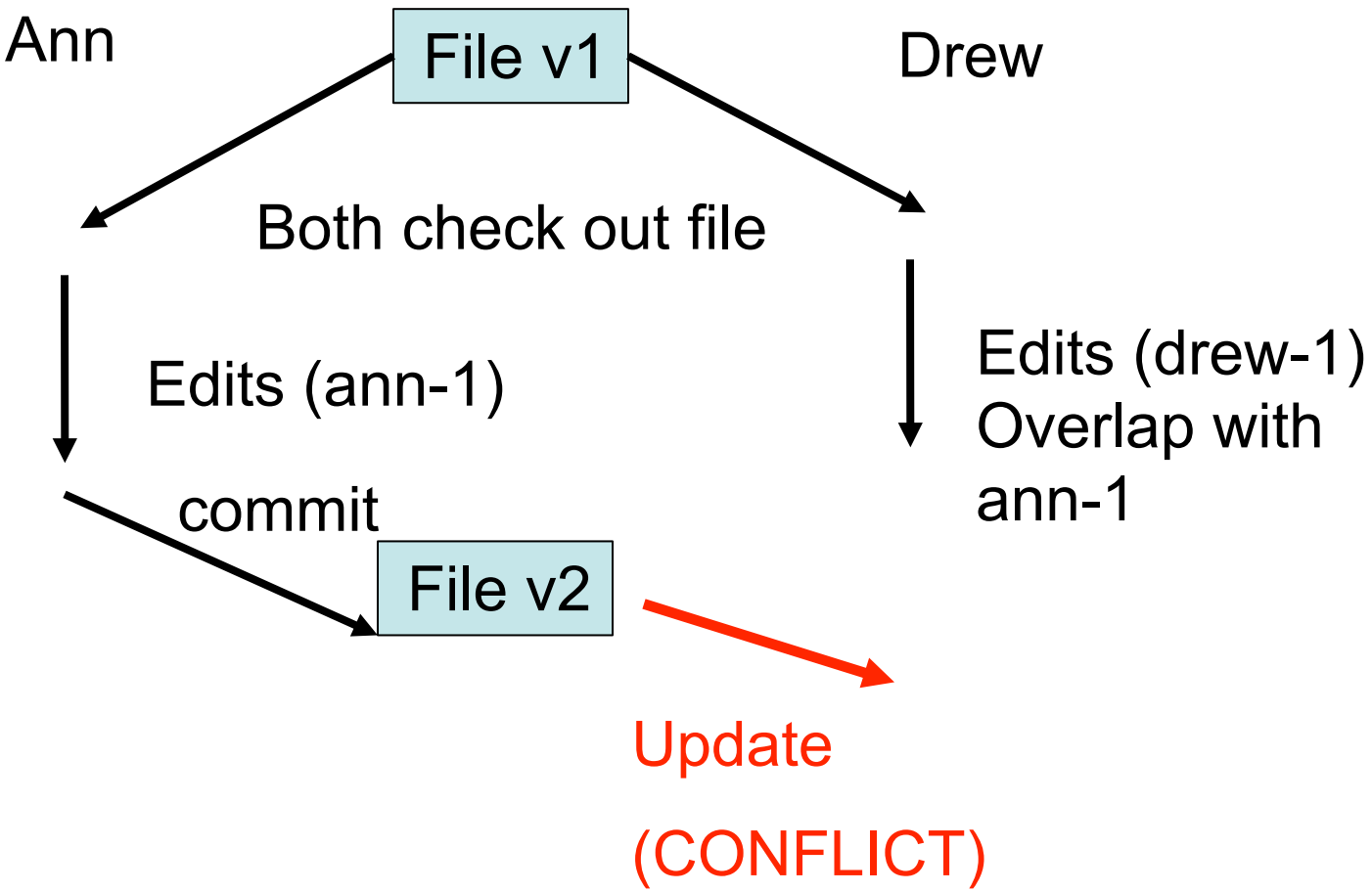
- Each person checks out a copy
- Both can work at the same time without (much) fear of clobbering the other
  - Changes only visible on commit/update

- What if both people edit the same file and commit it?

Concurrent edits



Concurrent edits



Resolving Conflicts

- Subversion will give you 3 files:
  - The original with conflict markers
  - The version you were editing
  - The latest version in the repository
- You can:
  - Keep your changes, discarding others
  - Toss your changes
  - Manually resolve

Branches

- Multiple paths of development, e.g.
  - Release 1.0 only gets security patches
  - “Development” branch gets everything
- “tags” or “snapshots”
  - Save a good known state. E.g., for handing in.
- Issue of merging (read on your own)

Subversion commands

- `svn checkout https://moo.cmcl.cs.cmu.edu/440/..`
- `svn commit`
- `svn update`

- `svn add`
- `svn mkdir`

- `svn copy:  create a branch or snapshot`
- `svn diff:  See difference between versions by default:  between what you started on and where you are now`

- **`svn help`**

Brief walkthrough

```
> svn checkout https://moo.cmcl.cs.cmu.edu/440/Project...
    A    trunk/
    …
> cd trunk
> echo “#empty Makefile” >> Makefile
> svn add Makefile
  A         Makefile
> svn commit
[svn will open an editor for log message]
Adding          Makefile
Transmitting file data ..
Committed revision 2.
```

Turning stuff in

```
> svn add server.cpp
    A    trunk/server.cpp
    …
# tested, it works!

> svn copy trunk tags/final
  A         tags/final
> cd tags; svn status
  A  +   final
# test your code in the final directory!
> svn commit
[svn will open an editor for log message]
...
Transmitting file data ..
```

Thoughts on Revision Control

- Update, make, test, *then* commit
- Update out of habit before you start editing
- Merge often
- Commit format changes separately
- Check *svn diff* before committing
- Try not to break the checked in copy
  - Invasive changes? Maybe a branch
- Don’t use svn lock
- Avoid conflicts by good decomposition (modularity) and out-of-band coordination

Go forth and revise!

- Revision control will save you untold pain
  - Most people I know have accidentally nuked files or entire directories
  - Logs and diffs very useful for finding bugs
  - Much better way to coordinate with partners (but useful on your own! We use it for almost everything)
- Very small investment to learn
- Try it on your own!
- Read the SVN book online for more info

A little story

Bob

Alice

> echo “TODO: Patch Alice’s box” > tempfiles/TODO
> cd bob_builder_files/
> vimacs patch.cpp
> cd ..
> rm -rf *files
...

Alice’s box gets compromised! :(

Bob

With svn

Eva

> echo “TODO: Patch Eva’s box” > tempfiles/TODO
> cd bob_builder_files/ <-- in subversion
> sed -i ‘s/Alice/Eva/g’ patch.cpp
> svn ci -m “modified patch to work with Eva architecture”
> cd .. && rm -rf *files
> svn co https://moo.cmcl.cs.cmu.edu/bob_builder_files/
> cd bob_builder_files && make && make test

Outline

- Resources
- Revision Control
- **Makefiles**
- General Q&A

Simple g++

If we have files:

- prog.cpp: The main program file
- lib.cpp: Library .cpp file
- lib.h: Library header file

% g++ -c prog.cpp -o prog.o

% g++ -c lib.cpp -o lib.o

% g++ lib.o prog.o -o binary

g++ flags

- Useful flags
  1. -g: debugging hook
  2. -Wall: all warning
  3. -Werror: treat warning as errors
  4. -O2, -O3: optimization
  5. -DDEBUG: macro for DEBUG (#define DEBUG)

Examples

```
% g++ -g -Wall -Werror -c prog.cpp -o prog.o
% g++ -g -Wall -Werror -c lib.cpp -o lib.o
% g++ -g -Wall -Werror lib.o prog.o -o binary
```

But Don’t Repeat Yourself!

Makefile

```
% g++ -g -Wall -Werror -c prog.cpp -o prog.o
% g++ -g -Wall -Werror -c lib.cpp -o lib.o
% g++ -g -Wall -Werror lib.o prog.o -o binary
```

```
CXX = g++
CFLAGS = -g -Wall -Werror
OUTPUT = binary
```

Makefile

```
target: dependency1 dependency2 ...
    unix command (start line with TAB)
    unix command
    ...

% g++ lib.o prog.o -o binary

binary: lib.o prog.o
    g++ lib.o prog.o -o binary
```

```
binary: lib.o prog.o
	g++ -g -Wall lib.o prog.o -o binary

lib.o: lib.cpp
	g++ -g -Wall -c lib.cpp -o lib.o

prog.o: prog.cpp
	g++ -g -Wall -c prog.cpp -o prog.o

clean:
	rm *.o binary
```

```
binary: lib.o prog.o
	g++ -g -Wall lib.o prog.o -o binary

lib.o: lib.cpp
	g++ -g -Wall -c lib.cpp -o lib.o

prog.o: prog.cpp
	g++ -g -Wall -c prog.cpp -o prog.o
```

```
CXX = g++
CXXFLAGS = -g -Wall
OUTPUT = binary

$(OUTPUT): lib.o prog.o
	$(CXX) $(CXXFLAGS) lib.o prog.o -o binary

lib.o: lib.cpp
	$(CXX) $(CXXFLAGS) -c lib.cpp -o lib.o

prog.o: prog.cpp
	$(CXX) $(CXXFLAGS) -c prog.cpp -o prog.o

clean:
```

```
CXX = g++
CXXFLAGS = -g -Wall
OUTPUT = binary

$(OUTPUT): lib.o prog.o
	$(CXX) $(CXXFLAGS) lib.o prog.o -o binary

lib.o: lib.cpp
	$(CXX) $(CXXFLAGS) -c lib.cpp -o lib.o

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

```
CXX = g++
CXXFLAGS = -g -Wall
OUTPUT = binary
OBJFILES = lib.o prog.o


$(OUTPUT): $(OBJFILES)
	$(CXX) $(CXXFLAGS) $(OBJFILES) -o binary

lib.o: lib.cpp
	$(CXX) $(CXXFLAGS) -c lib.cpp -o lib.o

prog.o: prog.cpp
	$(CXX) $(CXXFLAGS) -c prog.cpp -o prog.o

clean:
```

```
CXX = g++
CXXFLAGS = -g -Wall
OUTPUT = binary
OBJFILES = lib.o prog.o

$(OUTPUT): $(OBJFILES)
	$(CXX) $(CXXFLAGS) $(OBJFILES) -o binary

lib.o: lib.cpp
	$(CXX) $(CXXFLAGS) -c lib.cpp -o lib.o

prog.o: prog.cpp
	$(CXX) $(CXXFLAGS) -c prog.cpp -o prog.o

clean:
```

```
CXX = g++
CXXFLAGS = -g -Wall
OUTPUT = binary
OBJFILES = lib.o prog.o

$(OUTPUT): $(OBJFILES)
	$(CXX) $(CXXFLAGS) $(OBJFILES) -o binary

%.o: %.cpp
	# $<: dependency (%.cpp)
	# $@: target (%.o)
	$(CXX) $(CXXFLAGS) -c $< -o $@
```

Simple Test Script

```
% ./server 6667 &
% cat testfile.01 | ./testscript.py
% cat testfile.02 | ./testscript.py
% killall -9 server
```

Simple Test Script

```
#/bin/sh

echo “Starting server on port 6667.”
./server 6667 &
SERVERPID = $!

echo “Running test files.”
cat testfile.01 | ./testscript.py
cat testfile.02 | ./testscript.py

echo “Killing server process.”
kill $(SERVERPID)
```

```
CXX = g++
CXXFLAGS = -g -Wall
OUTPUT = binary
OBJFILES = lib.o prog.o


all: $(OUTPUT)

$(OUTPUT): $(OBJFILES)
	$(CXX) $(CXXFLAGS) $(OBJFILES) -o binary

%.o: %.cpp
	# $<: dependencies (%.cpp)
	# $@: target (%.o)
	$(CXX) $(CXXFLAGS) -c $< -o $@
```

```
CXX = g++
CXXFLAGS = -g -Wall
OUTPUT = binary
OBJFILES = lib.o prog.o


all: $(OUTPUT)  test

$(OUTPUT): $(OBJFILES)
	$(CXX) $(CXXFLAGS) $(OBJFILES) -o binary

%.o: %.cpp
	# $<: dependencies (%.cpp)
	# $@: target (%.o)
	$(CXX) $(CXXFLAGS) -c $< -o $@

test: $(OUTPUT)
	sh ./testscript.sh

clean:
	rm *.o $(OUTPUT)
```

Use Makefile

% make

% make test

% make clean

Google

- "makefile example"
- "makefile template"
- "make tutorial"

Project 1

- Stage 0
  - Create a Makefile that builds a “client” and “server” executable binary
  - Check in necessary files to your svn repository
  - Use svn copy to move into tags/stage0 dir!

Stage 0 testing

- mkdir tmp; cd tmp
- svn co repoURL
- cd Project1TeamX/tags/stage0
- make
  - (Makefile commands executed successflly)
- ls -lF
  - rwxr-xr-x 1 uid gid size date time server*
  - rwxr-xr-x 1 uid gid size date time client*
  - (other files)