CS35L – Spring 2018

Slide set:	3.1
Slide topics:	Modifying and Rewriting Software
Assignment:	3

How to Install Software

- Windows
  - Installshield
  - Microsoft/Windows Installer
- OS X
  - Drag and drop from .dmg mount -> Applications folder
- Linux
  - rpm(Redhat Package Management)
    - RedHat Linux (.rpm)
  - apt-get(Advanced Package Tool)
    - Debian Linux, Ubuntu Linux (.deb)
  - Good old build process
    - configure, make, make install

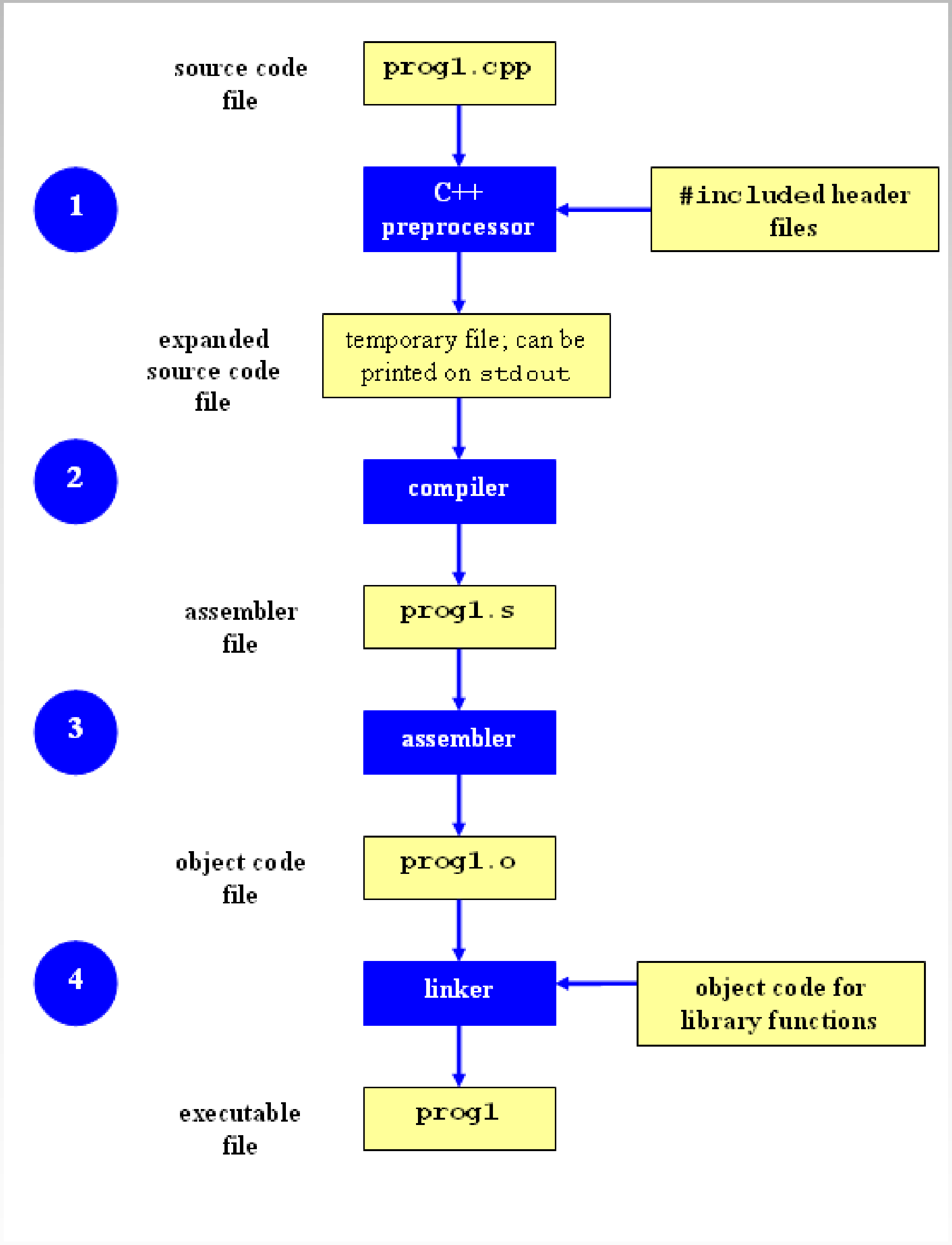
Decompressing Files

- Generally, you receive Linux software in the tarball format (.tgz) or (.gz)

Decompress file in current directory:

- $ tar –xzvf filename.tar.gz
  - Option –x: --extract
  - Option –z: --gzip
  - Option –v: --verbose
  - Option –f: --file

Compilation Process



Command-Line Compilation

- shop.cpp
  - #includes shoppingList.h and item.h
- shoppingList.cpp
  - #includes shoppingList.h
- item.cpp
  - #includes item.h
- How to compile?
  - **g++ -Wall shoppingList.cpp item.cpp shop.cpp –o shop**

What if…

- **We change one of the header or source files?**
  - Rerun command to generate new executable
- **We only made a small change to item.cpp?**
  - not efficient to recompile shoppinglist.cpp and shop.cpp
  - Solution: avoid waste by producing a separate object code file for each source file
    - g++ -Wall –c item.cpp… (for each source file)
    - g++ item.o shoppingList.o shop.o –o shop (combine)
    - Less work for compiler, saves time but more commands

What if…

- **We change item.h?**
  - Need to recompile every source file that includes it & every source file that includes a header that includes it. Here: item.cpp and shop.cpp
  - Difficult to keep track of files when project is large
    - Windows 7 ~40 million lines of code
    - Google ~2 billion lines of code

=> Make

Make

- Utility for managing large software projects
- Compiles files and keeps them up-to-date
- Efficient Compilation (only files that need to be recompiled)

Makefile Example

```
# Makefile - A Basic Example
all : shop  #usually first
shop : item.o shoppingList.o shop.o
        g++ -g -Wall -o shop item.o shoppingList.o shop.o
item.o : item.cpp item.h
        g++ -g -Wall -c item.cpp
shoppingList.o : shoppingList.cpp shoppingList.h
        g++ -g -Wall -c shoppingList.cpp
shop.o : shop.cpp item.h shoppingList.h
        g++ -g -Wall -c shop.cpp
clean :
        rm -f item.o shoppingList.o shop.o shop
```

Rule

🟩 Comments

🟦 Targets

🟪 Prerequisites

Dependency Line

🟫 Commands

Build Process

- **configure**
  - Script that checks details about the machine before installation
    - Dependency between packages
  - Creates ‘Makefile’
- **make**
  - Requires ‘Makefile’ to run
  - Compiles all the program code and creates executables in current temporary directory
- **make install**
  - make utility searches for a label named install within the Makefile, and executes only that section of it
  - executables are copied into the final directories (system directories)

Lab 3

- Coreutils 8.29 has a problem
  - $ la –A is equivalent to ls -a –A
  - if the current directory has two files named .foo and bar, the command la -A outputs four lines, one each for ., .., .foo, and bar.
  - These users want la -A to output just two lines instead, one for .foo and one for bar
- Why?
  - the -a option always overrides the -A option regardless of which option is given first
- Want the flag that comes later to take effect
- Fix the ls program

Getting Set Up (Step 1)

- Download coreutils-8.29 to your home directory
  - Use ‘wget’
- Untar and Unzip it
  - tar –xJvf coreutils-8.29.tar.xz
- Make a directory ~/coreutilsInstall in your home directory (this is where you’ll be installing coreutils)
  - mkdir ~/coreutilsInstall

Building coreutils (Step 2)

- Go into coreutils-8.29 directory. This is what you just unzipped.
- Read the INSTALL file on how to configure "make", especially **--prefix** flag
- Run the configure script using the prefix flag so that when everything is done, coreutils will be installed in the directory ~/coreutilsInstall
- Compile it: make
- Install it: make install (won't work on Linux server without proper prefix!)
  - Why?

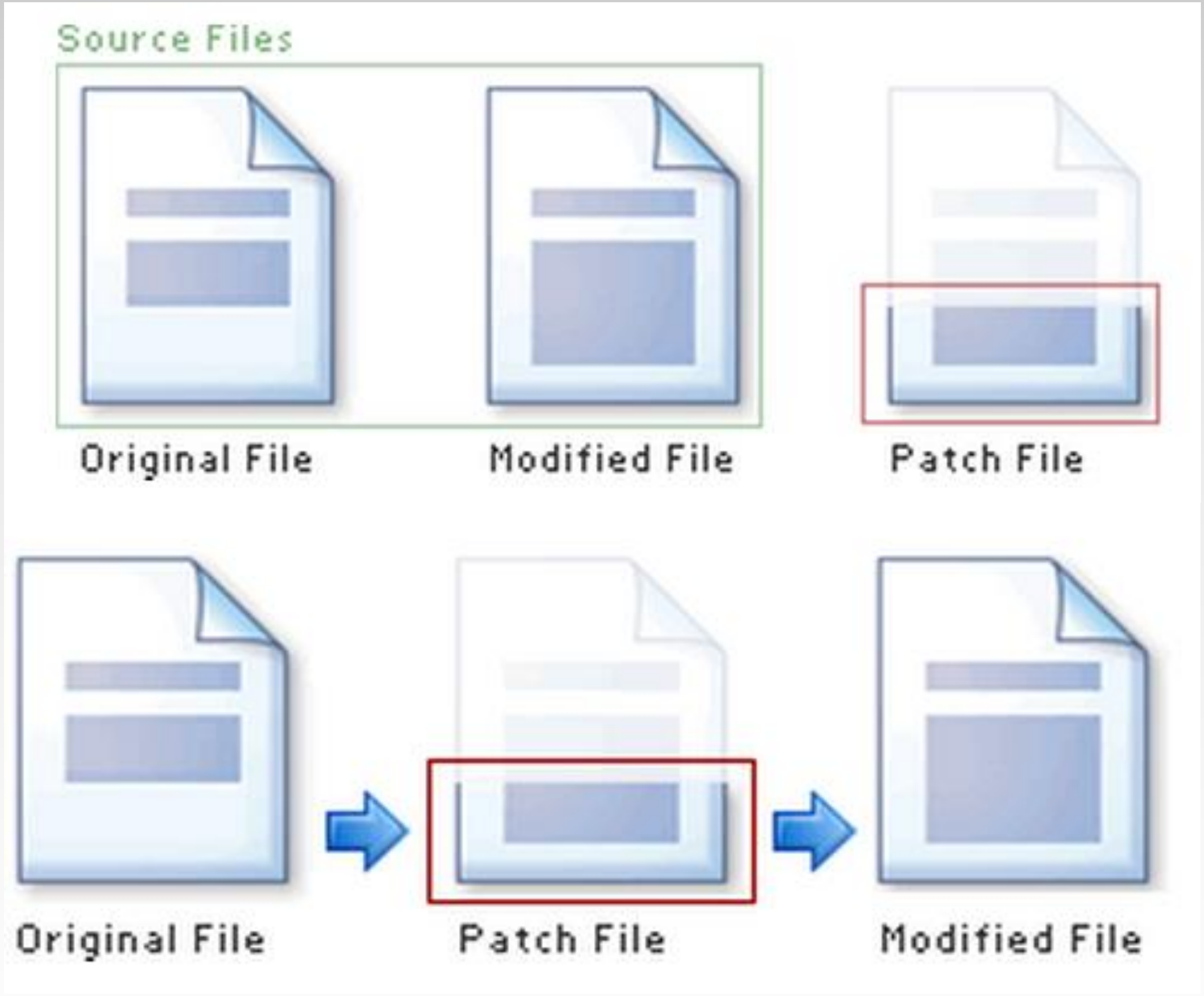
Reproduce Bug (Step 3)

- Reproduce the bug by running the version of ‘ls -a -A ’ in coreutils 8.29
- If you just type $ ls at CLI it won’t run ‘ls’ in coreutils 8.29
  - Why? Shell looks for /bin/ls
  - To use coreutils 8.29: $ ./ls
    - This manually runs the executable in this directory

Patching

- A patch is a piece of software designed to fix problems with or update a computer program
- It’s a diff file that includes the changes made to a file
- A person who has the original (buggy) file can use the patch command with the diff file to add the changes to their original file

Applying a Patch



diff Unified Format

- diff –u original_file modified_file

- --- path/to/original_file
- +++ path/to/modified_file

- @@ -l,s +l,s @@
  - @@: beginning of a hunk
  - l: beginning line number
  - s: number of lines the change hunk applies to for each file
  - A line with a:
    - - sign was deleted from the original
    - + sign was added to the original
    - stayed the same

Patching and Building (Steps 4 & 5)

- cd coreutils-8.29
- vim or emacs patch_file: copy and paste the patch content
- `patch` –**pnum** < patch_file
  - ‘`man patch`’ to find out what **pnum** does and how to use it
- `cd` into the coreutils-8.29 directory and type make to rebuild patched ls.c.
  - Don’t install!!

Testing Fix (Step 6)

- Test the following:
  - Modified ls works
  - Installed unmodified ls does NOT work
- Test on:
  - Empty directory
  - Directory containing a hidden file
    - With just –a, with just –A
    - With –aA
    - With –Aa
- Answer Q1 and Q2