Lecture 5

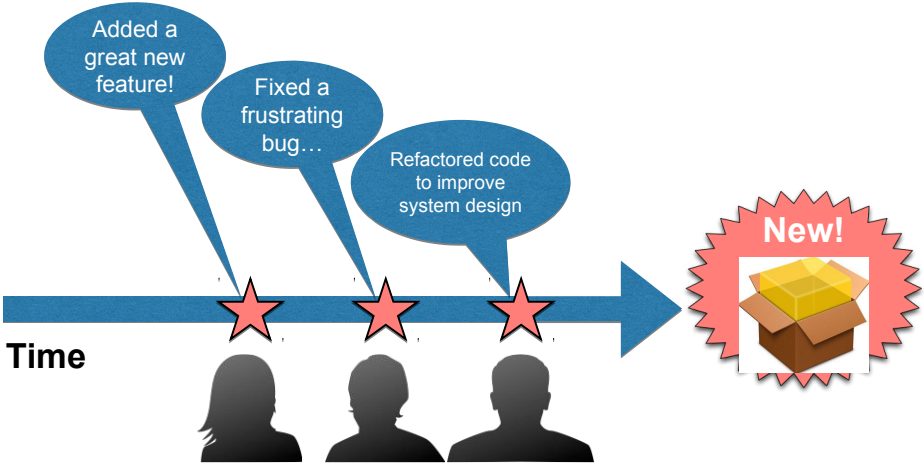
Build Systems

SOEN 6441, Summer 2018

René Witte
Department of Computer Science
and Software Engineering
Concordia University

Outline

What is a software release?



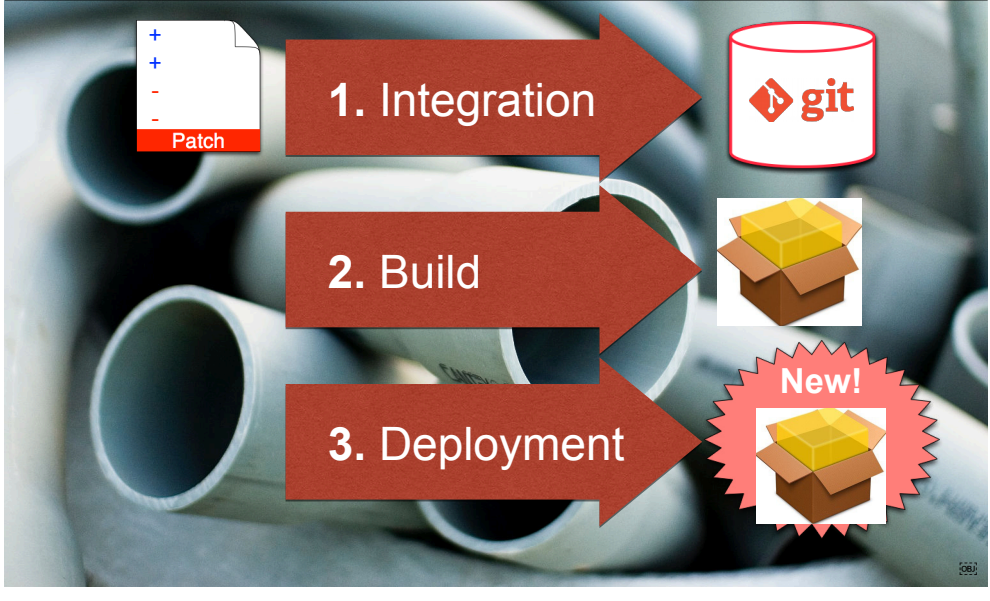
René Witte
Concordia University

The rapid release cycle of modern software systems

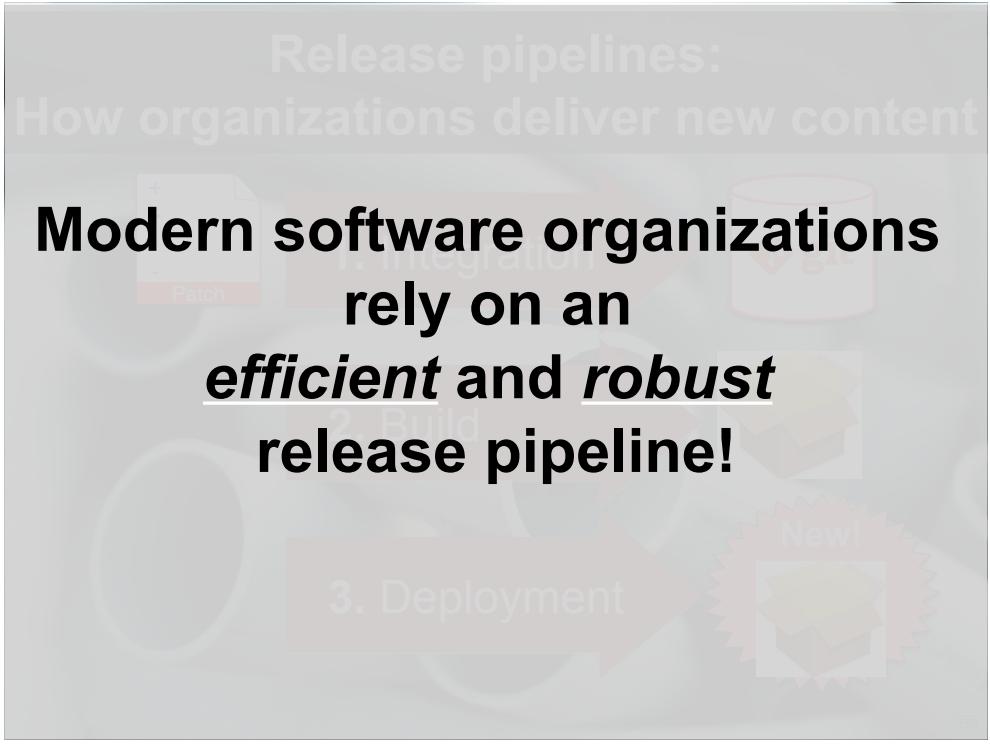
Often release several times in one day!

René Witte
Concordia University

Release pipelines: How organizations deliver new content



René Witte
Concordia University



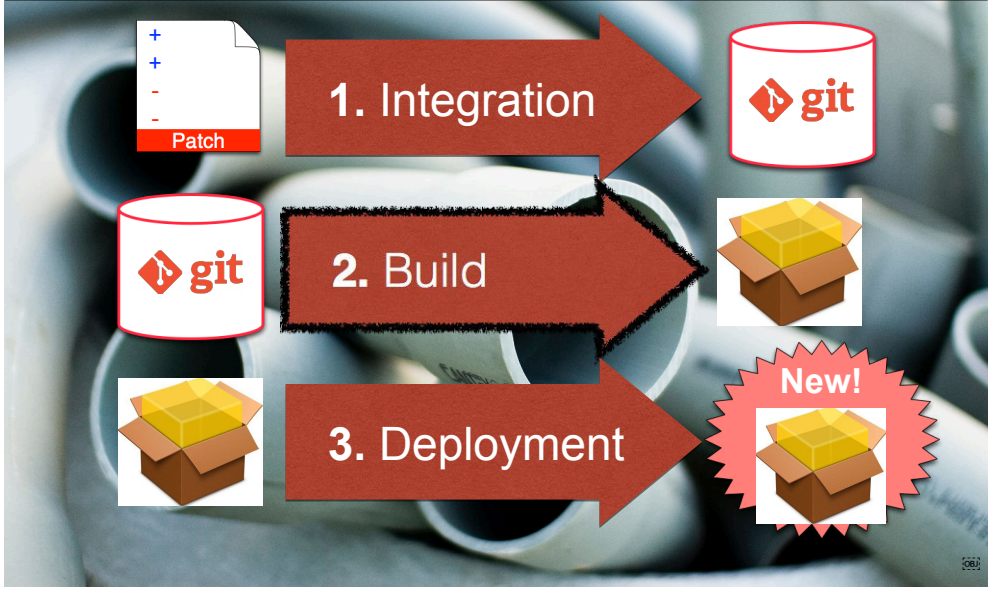
Modern software organizations rely on an *efficient* and *robust* release pipeline!

René Witte
Concordia University

Release engineers increase the market potential of software organizations

René Witte
Concordia University

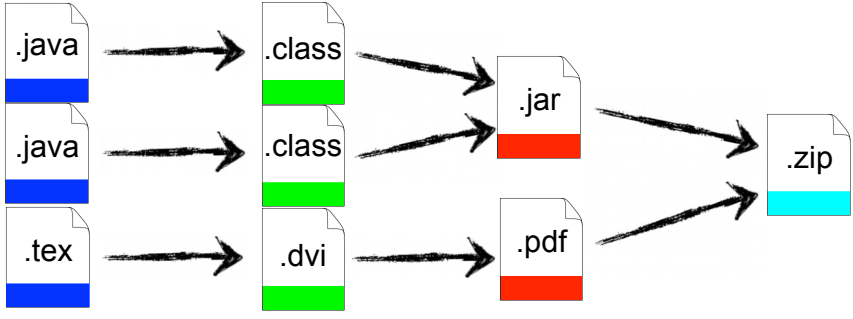
The rapid release cycle creates new software engineering challenges



René Witte
Concordia University

What is a build system?

René Witte
Concordia University



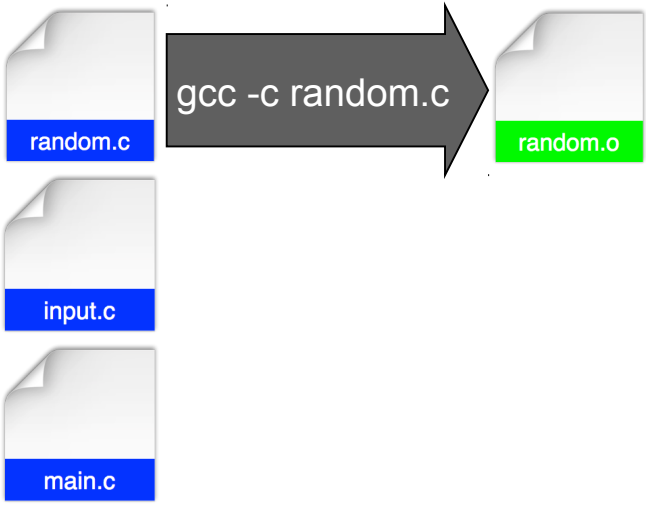
René Witte

Random number generating functions

User input handling functions

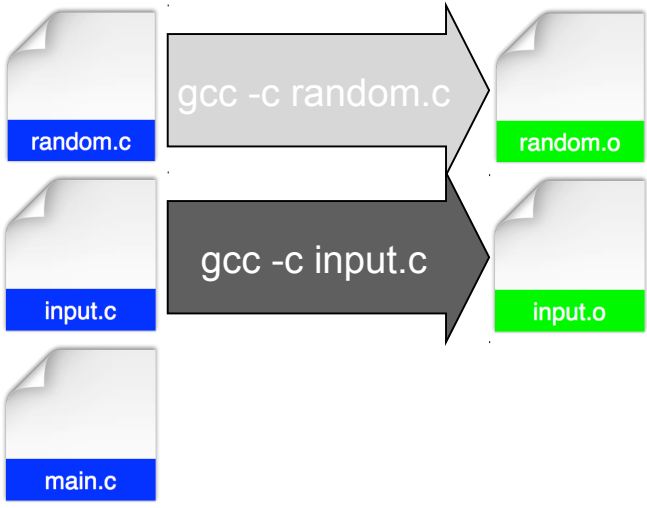
René Witte

Main function, calls input and random functions

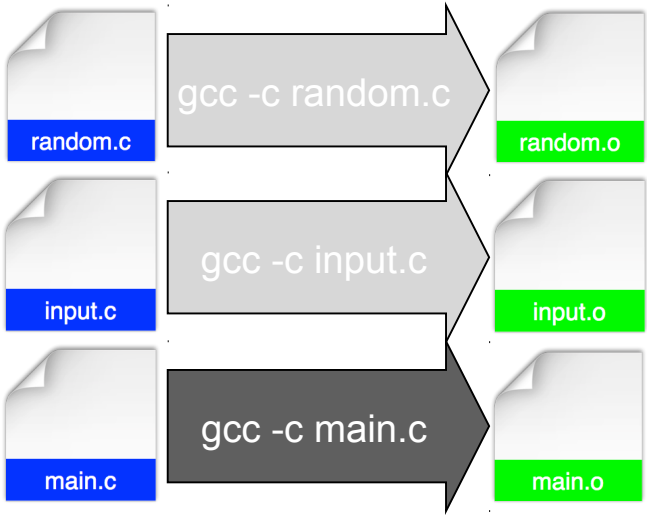


Compiling random.c

René Witte

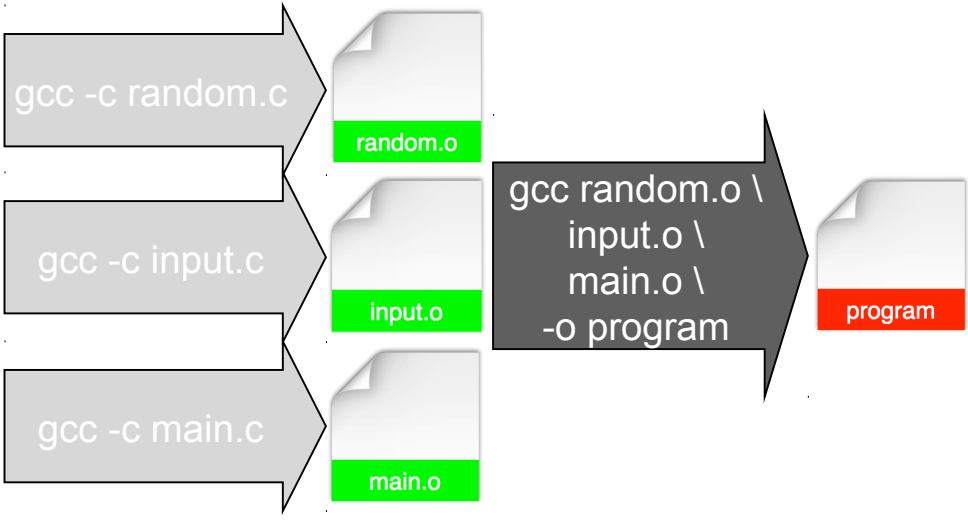


Compiling input.c



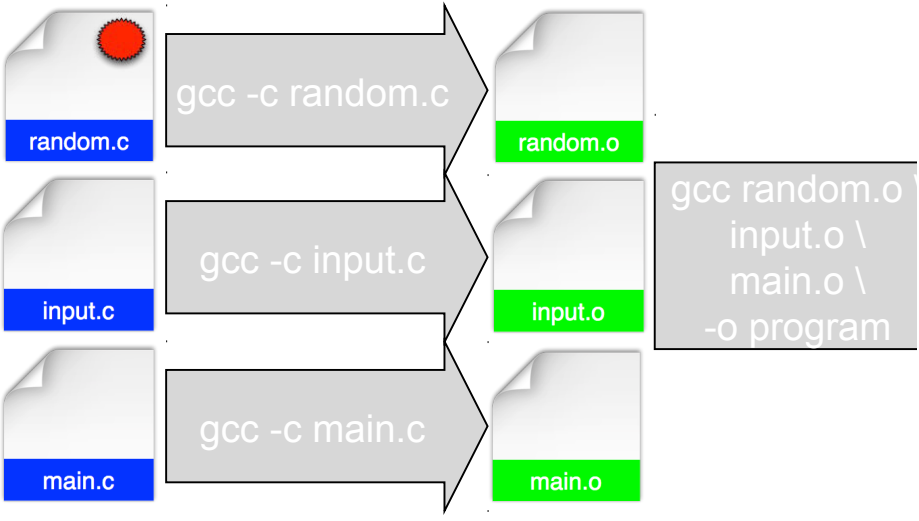
Compiling main.c

René Witte



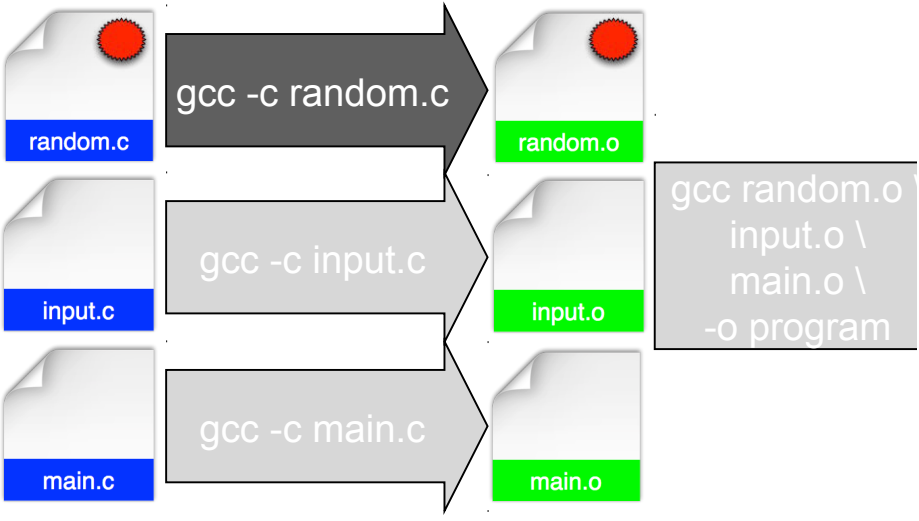
Link them together to complete the program

René Witte



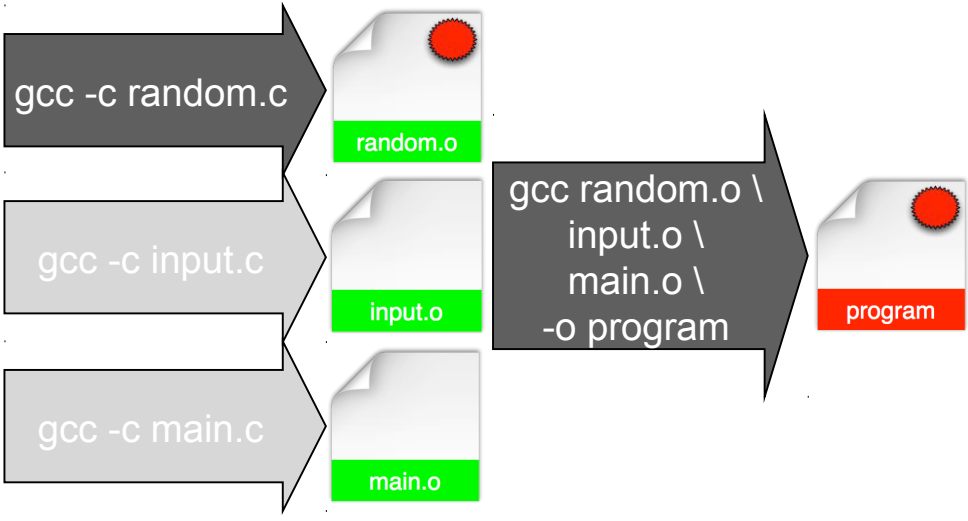
D’oh! We need to fix a bug in random.c!

René Witte



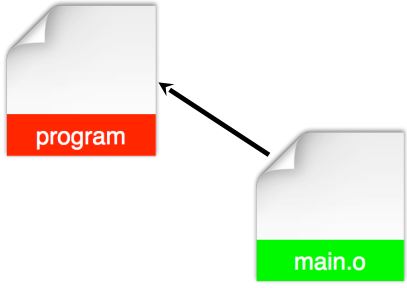
Recompile random.c

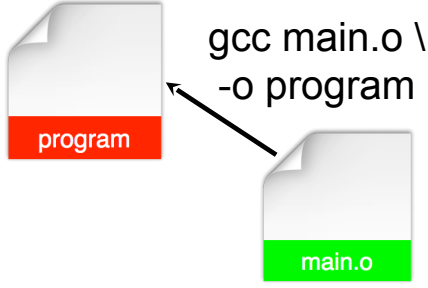
René Witte
Concordia University



Relink the objects together

make **to the rescue**



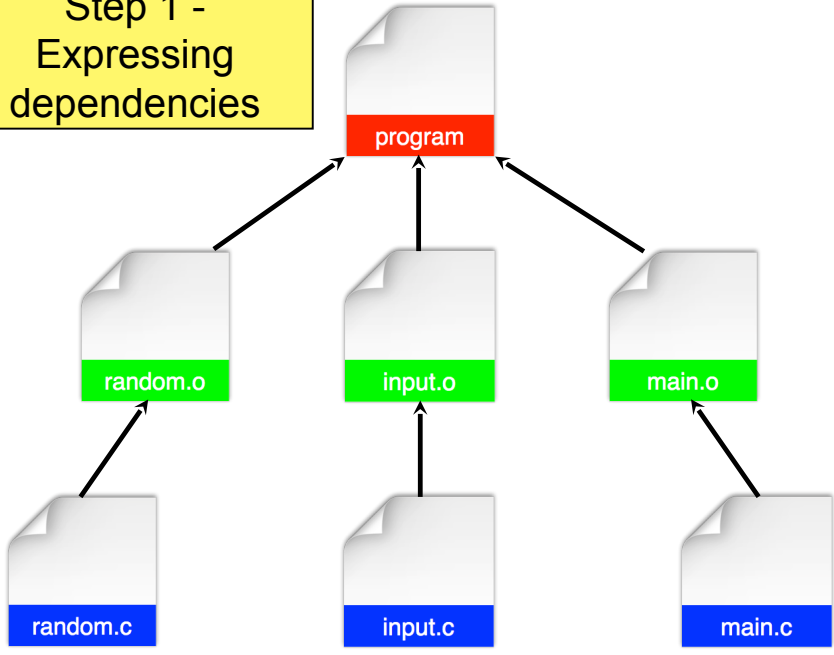


Step 1 - Expressing dependencies

Step 2 - Writing recipes

René Witte
Concordia University

Step 1 - Expressing dependencies

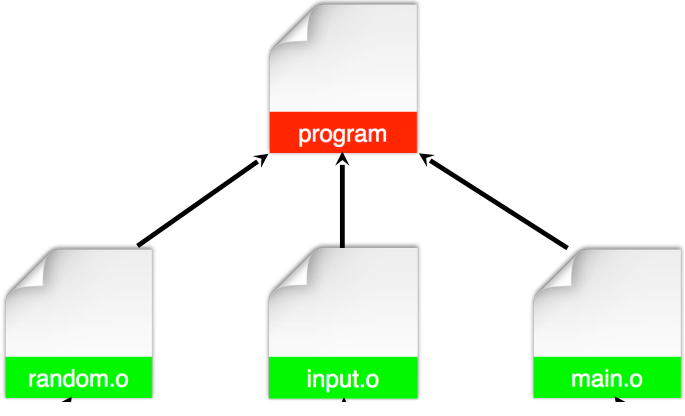


René Witte
Concordia University

Step 1 - Expressing dependencies

This is a **Makefile**

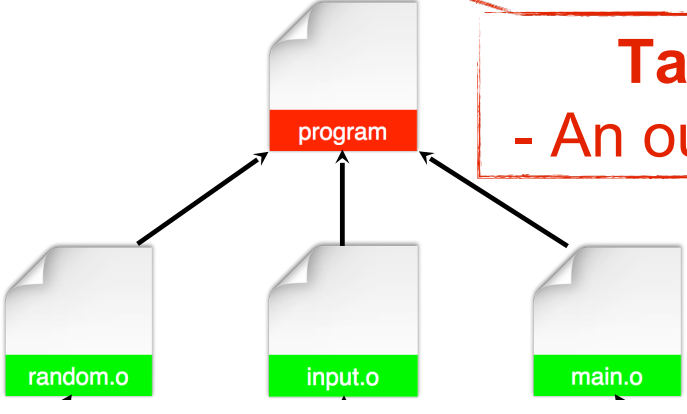
```
program : random.o input.o main.o
```



René Witte
Concordia University

Step 1 - Expressing dependencies

```
program : random.o input.o main.o
```

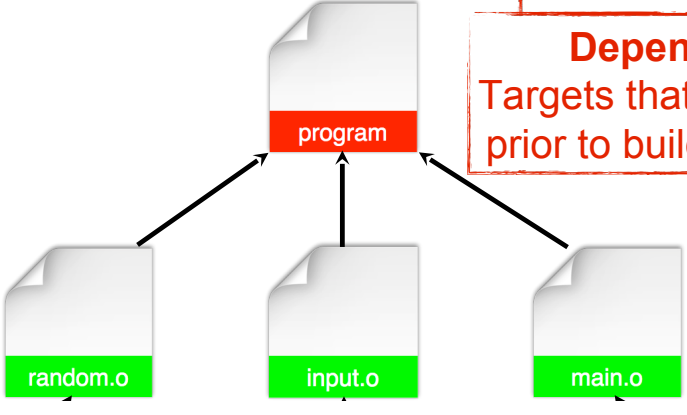


Target
- An output file

René Witte
Concordia University

Step 1 - Expressing dependencies

```
program : random.o input.o main.o
```

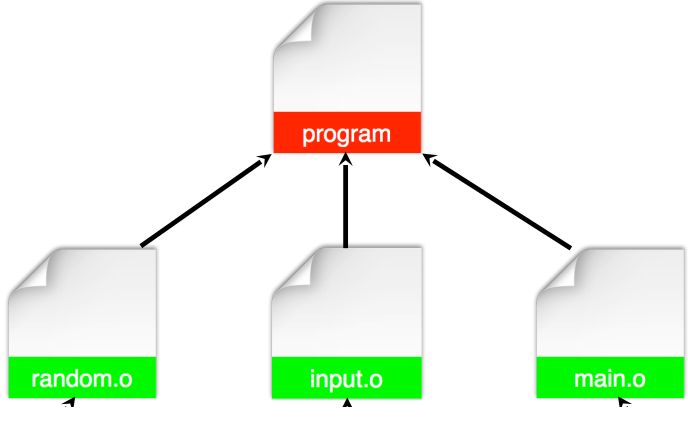


Dependencies
Targets that must be built prior to building this one

René Witte

Step 2 - Writing recipes

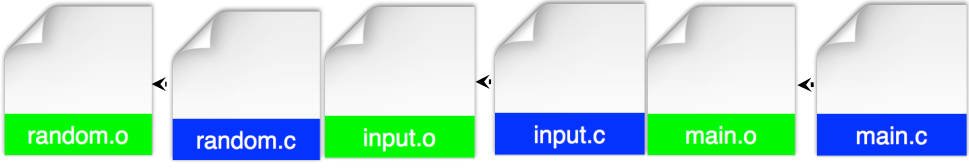
```
program : random.o input.o main.o
<tab>gcc -o program random.o input.o main.o
```



René Witte
Concordia University

Step 2 - Writing recipes

```
program : random.o input.o main.o
<tab>gcc -o program random.o input.o main.o
random.o : random.c
<tab>gcc -c random.c
input.o : input.c
<tab>gcc -c input.c
main.o : main.c
<tab>gcc -c main.c
```



Step 2 -
Writing recipes

Splitting long lines

```
program : random.o input.o main.o
<tab>gcc -o program random.o input.o main.o
```


```
program : random.o
program : input.o
program : main.o
program :
<tab>gcc -o program random.o input.o main.o
```

René Witte
Concordia University

Step 2 -
Writing recipes

Merging short lines

```
random.o : random.c
<tab>gcc -c random.c
```


```
random.o : random.c ; gcc -c random.c
```

René Witte
Concordia University

Variables

```
my_command = “gcc -c random.c”
random.o : random.c
<tab>$(my_command)
```

Variable assignment

```
my_filename = “random.c”

#Lazy set
my_command = “gcc -c $(my_filename)”
#Immediate set
my_command :=  “gcc -c $(my_filename)”
#Set if absent
my_command ?= “gcc -c $(my_filename)”
#Append
my_command += “gcc -c $(my_filename)”
```

René Witte
Concordia University

Watch out for lazy evaluation!

```
my_filename = “random.c”
my_command = “gcc -c $(my_filename)”
my_filename = “another_file.c”

random.o : random.c
<tab>$(my_command)
```

When random.o is built, the my_command is gcc -c another_file.c

Immidiate set

```
my_filename = “random.c”
my_command := “gcc -c $(my_filename)”
my_filename = “another_file.c”# this line
#does not impact $(my_command)

random.o : random.c
<tab>$(my_command)
```

Generates: gcc –c random.c

René Witte

Remember our Makefile?

```
program : random.o input.o main.o
<tab>gcc -o program random.o input.o main.o

random.o : random.c
<tab>gcc -c random.c

input.o : input.c
<tab>gcc -c input.c

main.o : main.c
<tab>gcc -c main.c
```

René Witte
Concordia University

make **command reads the** makefile

Terminal

```
$ make
gcc -c random.c
gcc -c input.c
gcc -c main.c
gcc -o program random.o input.o main.o
$
```

Rebuilds first target expressed in the makefile

René Witte
Concordia University

Alternatively…

Terminal

```
$ rm program random.o input.o main.o
$ make random.o
gcc -c random.c
$
```

Specify a target to be built

Alternatively…

Terminal

```
$ rm program random.o input.o main.o
$ make -f my_makefile
gcc -c random.c
gcc -c input.c
gcc -c main.c
gcc -o program random.o input.o main.o
$
```

Use -f flag to read from other files

René Witte
Concordia University

make Incremental Builds!

Terminal

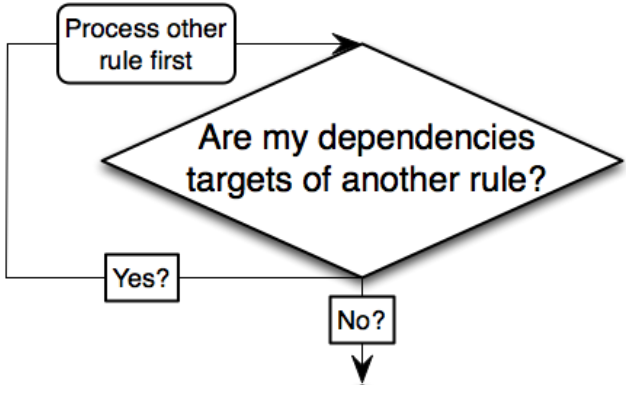
```
$ vim random.c
...
$ make
gcc -c random.c
gcc -o program random.o input.o main.o
$
```

Only the out-of-date targets are rebuilt!

René Witte
Concordia University

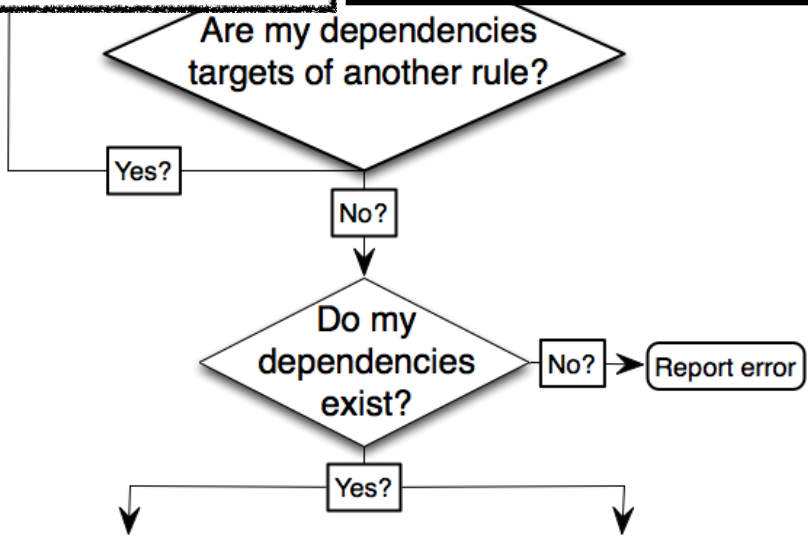
How does make work?

```
program : main.o
<tab>gcc -o program main.o
main.o : main.c
<tab>gcc -c main.c
```

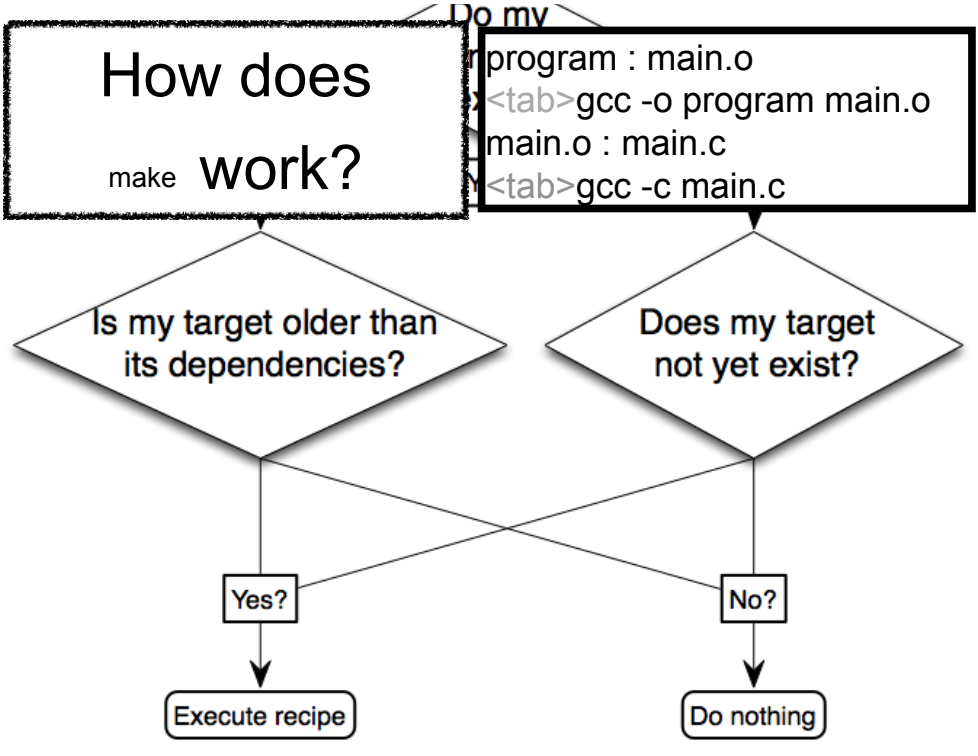


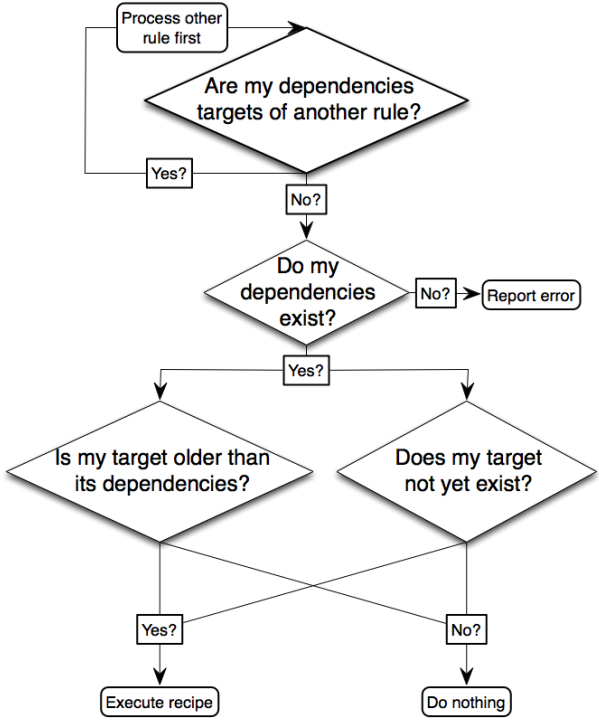
How does make work?

```
program : main.o
<tab>gcc -o program main.o
main.o : main.c
<tab>gcc -c main.c
```



René Witte

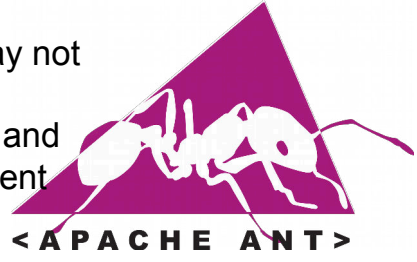

How does
make work?
program : main.o
<tab>gcc -o program main.o
main.o : main.c
<tab>gcc -c main.c
Is my target older than its dependencies?
Does my target not yet exist?
Yes?
No?
Execute recipe
Do nothing


Process other rule first
Are my dependencies targets of another rule?
Yes?
No?
Do my dependencies exist?
No?
Report error
Yes?
Is my target older than its dependencies?
Does my target not yet exist?
Yes?
No?
Execute recipe
Do nothing

Introduction to Apache Ant

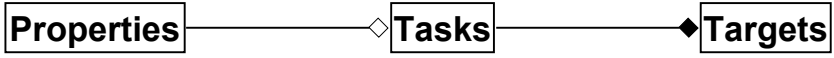
A make **replacement for Java systems:**

- Java compiler may recompile several files at once!
  - For example, javac foo.java may recompile bar.java, too!
  - The make tool is not good at recognizing side effects of commands like this
- Make recipes are written in platform-specific shell scripts
- If a recipe works on OS X, it may not
  - work on windows
  - Ant tasks are Java programs and
  - are mostly platform independent



René Witte
Concordia University

Key Ant programming concepts

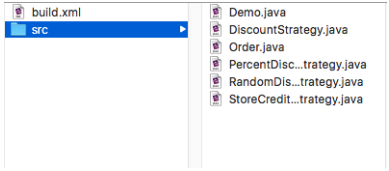


- Store and retrieve frequently used values
- Like variables in Java

- Correspond to commands that are executed
- Like issuing a call to a library or Java API

- A grouping of tasks that relate to a logical event
- Like grouping statements into methods in Java

Let’s see an Ant system!



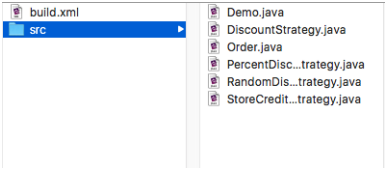
- Compile everything:

```
<target name=“compile”>
        <javac srcdir="src" destdir="${classdir}" />
</target>
```

- Specifying output directory

```
<target name=“init”>
<mkdir dir=“${blddir}” />
<mkdir dir=“${classdir}” />
</target>
```

Let’s see an Ant system!



- Compile everything:

```
<target name=“compile”>
  <javac srcdir=“src” destdir=“${classdir}” />
</target>
```

- Specifying output directory

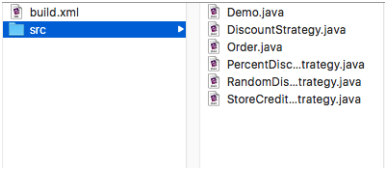
```
<target name=“init”>
<mkdir dir=“${blddir}” />
<mkdir dir=“${classdir}” />
</target>
```

- Specifying properties

```
<property name="blddir" location="build" />  <property name="classdir" location="${blddir}/classes" />
</target>
```

René Witte
Concordia University

Let’s see an Ant system!



- Compile everything:

```
<target name=“compile”>
  <javac  srcdir=“src”
destdir=“${classdir}”
depends="init" />
</target>
```

- Specifying output directory

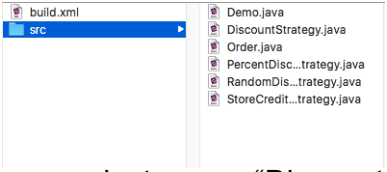
```
 <target name=“init”>
<mkdir dir=“${blddir}” />
<mkdir dir=“${classdir}” />
</target>
```

- Specifying properties

```
 <property name="blddir"
location="build" />  <property
name="classdir" location="$
{blddir}/classes" />
</target>
```

René Witte

Let’s see an Ant system!




```
<project name=“DiscountStrategy” default=“compile”>
    <property name=“blddir” location=“build” />
    <property name=“classdir” location=“${blddir}/classes” />
<target name=“init”>
    <mkdir dir=“${blddir}” />
    <mkdir dir=“${classdir}” />
</target>
<target name=“compile” depends=“init”>
    <javac      srcdir=“src”      destdir=“${classdir}” />
</target>
</project>
```

René Witte
Concordia University

Modern Build Systems

Make/Makefiles
- 1976
- (additional tools for Makefile generation etc: automake, iMake, ...)

Apache Ant:
- 2000

Apache Ivy:
- 2004
- Dependency management for ant

Apache Maven:
- 2004
- Project and dependency management

Gradle:
- 2007

Platform-specific tools
- sbt (Scala), rake (Ruby), Grunt (Javascript), buildr, Perforce Jam, ...

René Witte
Concordia University

Apache Ivy

René Witte

Dependency Management

- Your great new software needs library X
- (possibly in a specific major/minor version)
- but library X also needs library Y...

Automatically resolving these dependencies is an important part of modern software development

Apache Ivy to the rescue!

Apache Ivy (http://ant.apache.org/ivy/) is a sub-project of ant:

- Managing project dependencies
- XML-driven declaration of project dependencies and JAR repositories
- Automatic retrieval of transitive dependency definitions and resources
- Automatic integration to publicly available artifact repositories
- Resolution of dependency closures
- Configurable project state definitions, which allow for multiple dependency-set definitions
- Publishing of artifacts into a local enterprise repository

Ivy Example

René Witte
Concordia University

ivy.xml

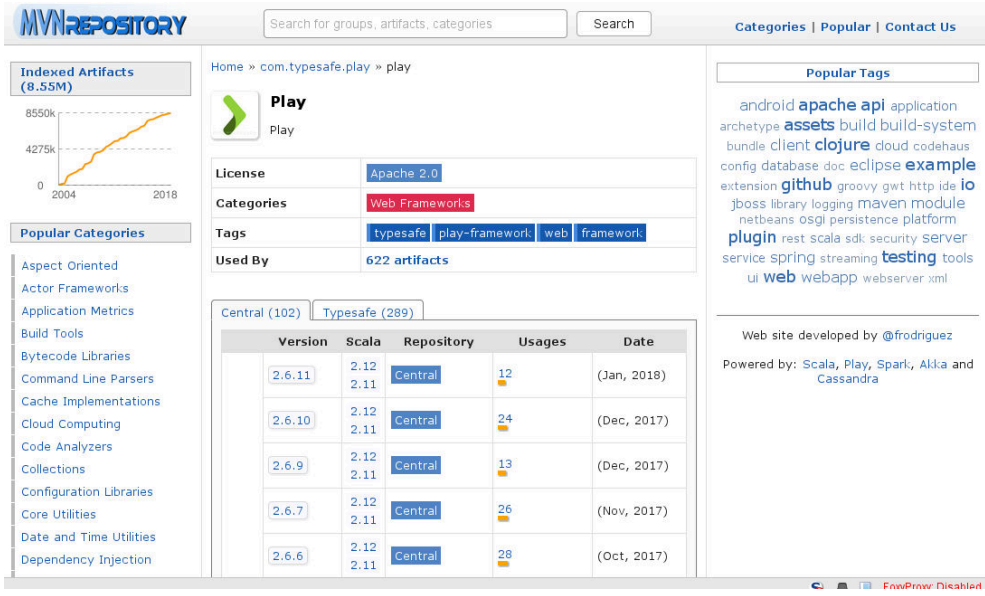
```
<ivy-module version="2.0">
  <info organisation="org.apache" module="hello-ivy"/>
  <dependencies>
    <dependency org="commons-lang" name="commons-lang" rev="2.0"/>
    <dependency org="commons-cli" name="commons-cli" rev="1.0"/>
  </dependencies>
</ivy-module>
```

build.xml

```
<project xmlns:ivy="antlib:org.apache.ivy.ant" name="hello-ivy" default="run">
    ...
    <!-- ================================
          target: resolve
         ================================ -->
    <target name="resolve" description="--> retrieve dependencies with ivy">
        <ivy:retrieve />
    </target>
</project>
```

Note: version numbers generally follow Semantic Versioning rules, see https://semver.org/

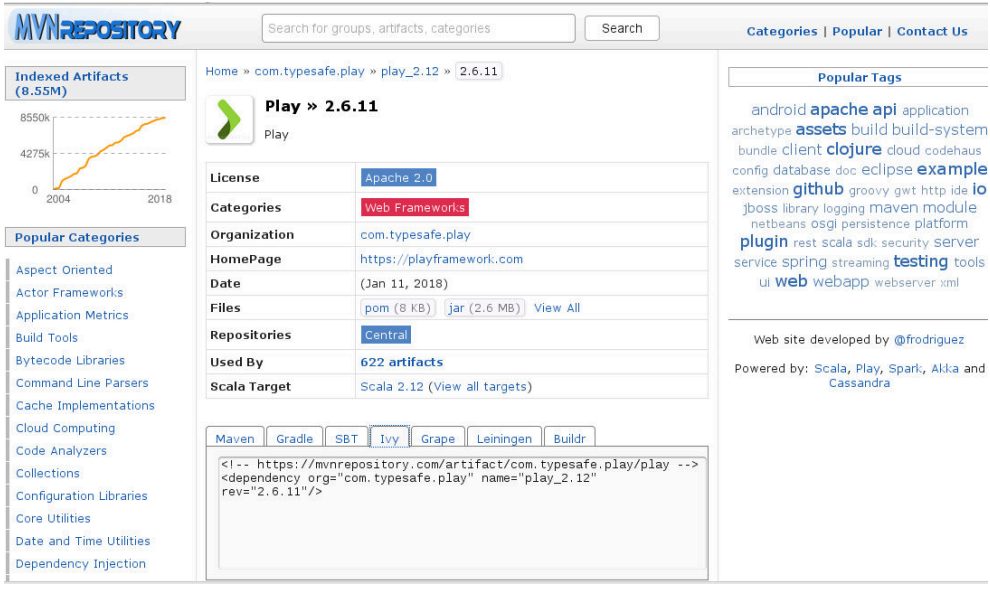
Maven Central, https://mvnrepository.com/repos/central



René Witte
Concordia University

Play in Maven Central

René Witte



Outline

Reading Material

Supplementary

- [Som16, Chapter 25] (Configuration Management)

References

[Som16] Ian Sommerville.
Software Engineering.
Pearson, 10th edition, 2016.
http://software-engineering-book.com.