

The image features three overlapping circles of varying sizes, drawn with thin, light gray lines. The circles are arranged in a way that they overlap each other, with the largest circle on the right and the two smaller ones on the left. In the center of the overlapping area, the word "Ant" is written in a bold, black, sans-serif font.

**Ant**

- Ferramenta para construção automática de *builds*
- Características:
  - *Open source*/Java
  - Facilmente extensível
  - *Build scripts* escritos em XML
  - Integração com **diversas ferramentas de desenvolvimento** e gerência de configuração

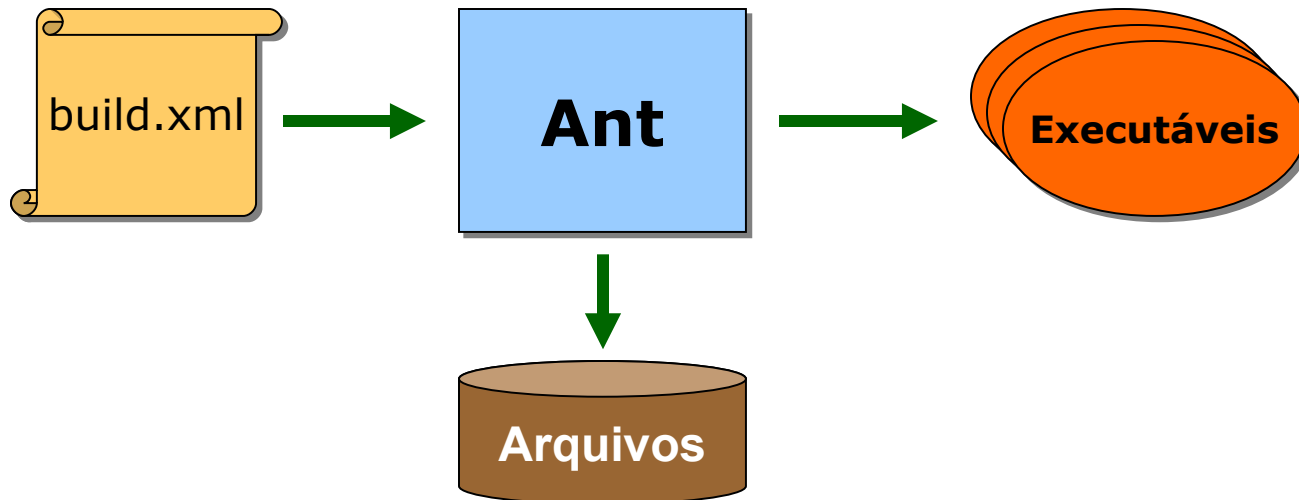
# O que o Ant pode fazer por você...

---

- Copiar, apagar, mover arquivos e diretórios
- Acessar o seu repositório
- Compilar os arquivos fontes
- Gerar um JAR
- Executar uma bateria de testes
- Enviar um e-mail com os resultados dos testes
- Fazer um *upload* do build via FTP para o servidor de produção

# Buildfiles

- **Buildfiles** são *scripts* a partir dos quais o Ant gera *builds*
- Escritos em XML



# Estrutura de um buildfile

```
<project>
```

O buildfile é composto de um projeto

```
  <property 1/>
```

```
    ⋮
```

```
  <property n/>
```

O projeto pode possuir propriedades

```
  <target 1>
```

```
    <task 1/>
```

```
  </target>
```

O projeto possui um conjunto de alvos

```
    ⋮
```

```
  <target n>
```

```
    <task 1/>
```

```
    <task n/>
```

Cada alvo é formado por tarefas

```
  </target>
```

```
</project>
```

# Projetos (*projects*)

---

- Elemento mais externo de um *buildfile*
- Cada projeto contém um conjunto de **alvos**
- Um projeto tem os seguintes atributos:

Atributo	Obrigatório
<b>name</b>	Não
<b>default</b>	Sim
<b>basedir</b>	Não

# Projetos - Exemplo

---

```
<project name="Projeto1" default="dist" basedir=". ">
```

```
    (tarefas agrupadas em alvos...)
```

```
</project>
```

# Olá Mundo

---

```
<?xml version="1.0" encoding="UTF-8" ?>
<project name="ProjOla" default="ola" basedir=".">

  <target name="ola" >
    <echo message="Ola, Ant trabalhando!" />
  </target>

</project>
```



# Propriedades (*properties*)

---

- Um projeto pode ter um conjunto de propriedades
- Essas propriedades podem ser criadas através da tarefa **property** ou fora do Ant
- Propriedades podem ser usadas como valores para atributos de tarefas
- Isso é feito colocando-se o nome da propriedade entre "\$ {" e "}" no valor do atributo

# Propriedades - Exemplos

---

## Definição

```
<property file="build.properties"/>
```

```
<property name="src" value="."/>
```

```
<property name="dist" value="distribuicao"/>
```

```
<property name="build" value="build"/>
```

## Uso

```
<javac srcdir="${src}" destdir="${build}"/>
```

```
<mkdir dir="${dist}/lib"/>
```

```
<jar jarfile="${dist}/lib/build.jar"  
  basedir="${build}"/>
```

# Propriedades Pré-Definidas

---

- Ant permite acesso às **propriedades do sistema operacional**
  - Funciona como se as propriedades tivessem sido definidas com uma tarefa **property**
  - As propriedades disponíveis estão listadas no javadoc de **System.getProperties**
- Além dessas, Ant possui algumas propriedades pré-definidas:

• <b>basedir</b>	• <b>ant.project.name</b>
• <b>ant.file</b>	• <b>ant.java.version</b>
• <b>ant.version</b>	

# Alvos (*targets*)

---

- Um alvo é um conjunto de tarefas que se quer executar
- Representa um objetivo a ser alcançado
- Exemplos:
  - Criação dos diretórios usados no processo de geração do *build*
  - Compilação dos arquivos-fonte
  - Empacotamento dos arquivos compilados
  - Remoção de arquivos temporários

# Dependências entre Alvos

---

- Um alvo pode depender de outros alvos
- Se um alvo **B** depende de outro alvo, **A**, **A** será processado antes de **B**
- Uma dependência é indicada através do atributo **depends** de um alvo
- Exemplo:

```
<target name="A" />
```

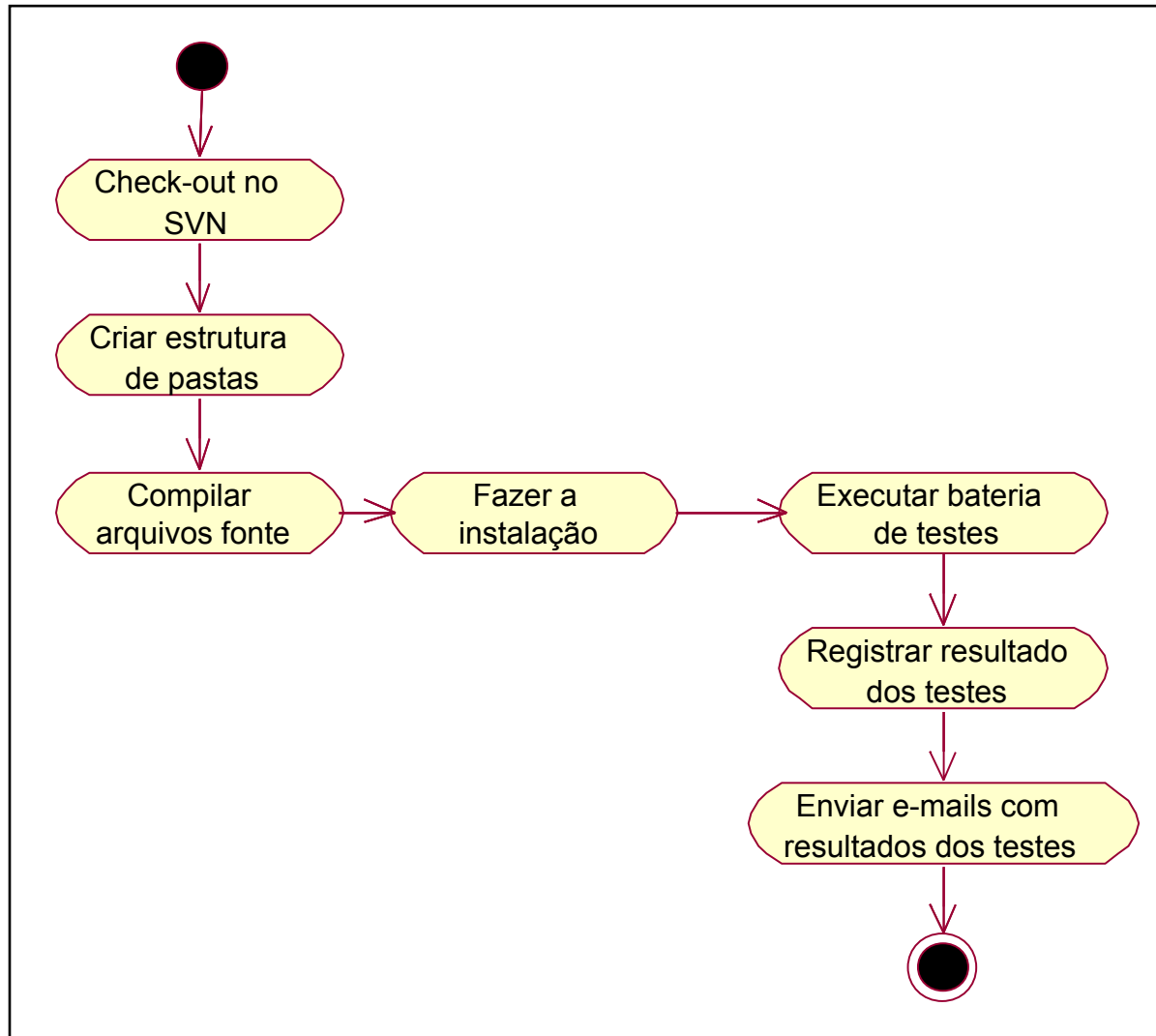
```
<target name="B" depends="A" />
```

```
<target name="C" depends="B" />
```

```
<target name="D" depends="C, B, A" />
```

Cada alvo é executado apenas uma vez!

# Exemplo



# Execução Condicional de Alvos

---

- É possível especificar que um alvo só será executado se:
  - Uma propriedade estiver setada
  - Uma propriedade **não** estiver setada
- Exemplos:

```
<target name="construa_o_modulo_A"  
  if="modulo_A_presente">
```

```
<target name="construa_o_modulo_A_falso"  
  unless="modulo_A_presente">
```

# Atributos de um Alvo

---

- Um alvo tem os seguintes atributos:

Atributo	Obrigatório
<b>name</b>	Sim
<b>depends</b>	Não
<b>if</b>	Não
<b>unless</b>	Não
<b>description</b>	Não



# Tarefas (*tasks*)

---

- Um pedaço de código que pode ser executado
- Unidades básicas de um *buildfile*
- Uma tarefa pode ter diversos atributos
- A distribuição do Ant já inclui um conjunto básico de tarefas
- Um pacote com tarefas adicionais também está disponível

# Tarefas - Exemplos

---

- **copy** - Copia um arquivo ou conjunto de arquivos
- **jar** - Empacota um conjunto de arquivos em um arquivo **.jar**
- **cvs** - Executa comandos do CVS
- **javac** - Compila um conjunto de arquivos fontes Java
- **mkdir** - Cria um novo diretório

# Tarefa `timestamp`

---

- Guarda informação temporal referente a quando a tarefa foi executada
  - **DSTAMP** - Formato `aaaammdd`
  - **TSTAMP** - Formato `hhmm`
  - **TODAY** - Formato `mês dia ano`
  - Permite formatação da data
- Exemplo

`<timestamp/>`

# Tarefa copy

---

- Copia arquivos e diretórios
- Um arquivo é copiado apenas se o arquivo fonte é mais novo que o destino (caso este último exista)
  - Sobre-escrita pode ser explicitada

# Tarefa copy (exemplos)

---

1. `<copy file="meuArq.txt" tofile="minhaCopia.txt"/>`
2. `<copy file="meuArq.txt" todir="../algum/dir"/>`
3. `<copy todir="../novo/dir">  
 <fileset dir="dir_fonte"/>  
</copy>`
4. `<copy todir="../novo/dir">  
 <fileset dir="${dist}" casesensitive="yes">  
 <patternset>  
 <include name = "**/*.java" />  
 <exclude name = "**/*Test*" />  
 </patternset>  
 </fileset>  
</copy>`

# Tarefa delete

---

- Apaga arquivos e diretórios
- Exemplos:

1. `<delete file="/lib/ant.jar"/>`

2. `<delete dir="lib"/>`

3. `<delete>`

`<fileset dir="." includes="**/*.bak"/>`  
`</delete>`

4. `<delete includeEmptyDirs="true" >`

`<fileset dir="build" />`  
`</delete>`

# Tarefa echo

---

- Escreve uma mensagem no console do usuário ou em um arquivo
- Exemplos:

1. `<echo message="Alo mundo!" />`

2. `<echo>`

Essa mensagem é bem maior, ocupando mais de uma linha.

`</echo>`

3. `<echo file="build.log" append="true">`

Esta mensagem será escrita em um arquivo!

`</echo>`

# Tarefa cvs

---

- Lida com módulos contidos em um repositório CVS
- Exemplos:
  1. `<cvs cvsRoot=":local:/arquivos/cvspublico" package="qib" dest="${build}" />`
  2. `<cvs dest="${dist}" command="update"/>`
  3. `<cvs command="-q diff -u -N" output="difs.txt"/>`
  4. `<cvs command="update -A -d"/>`



# Tarefas `mkdir` e `mail`

---

- `mkdir` - Cria uma árvore de diretórios

```
<mkdir dir="${dist}/lib/temp"/>
```

- `mail` - Manda uma mensagem de *e-mail* via SMTP

```
<mail from="suporte@cin.ufpe.br"  
  tolist="admin@cin.ufpe.br"  
  subject="Resultados do build noturno"  
  files="build.log"/>
```

# Tarefa javac

---

- Compila uma árvore de arquivos fonte Java
- É possível escolher um compilador diferente através da propriedade **build.compiler**

- **classic**

- **modern**

- **jikes**

- **jvc**

- **gcj**

- **sj**

- *Filesets* podem ser usados para definir os alvos da compilação

# Tarefa javac (exemplos)

---

1. `<javac srcdir="${src}" destdir="${build}"  
classpath="bib.jar" />`
2. `<javac srcdir="${src}" destdir="${build}"  
includes="meupacote/p1/**, meupacote/p2/**"  
excludes="meupacote/p1/pacotetestes/**"  
classpath="bib.jar" />`
3. `<javac destdir="${build}" classpath="bib.jar">  
 <src path="${src}" />  
 <src path="${src2}" />  
 <include name="meupacote/p1/**" />  
 <include name="meupacote/p2/**" />  
 <exclude name="meupacote/p1/pacotetestes/**" />  
</javac>`

# Tarefa java

---

- Executa uma classe Java na máquina virtual atual ou inicia outra, se especificado
- Exemplos:

1. `<java classname="teste.Principal" />`

2. `<java classname="teste.Principal" >  
 <arg value="-h"/>  
 <classpath>  
 <pathelement location="\test.jar" />  
 <pathelement path="${java.class.path}" />  
  
 </classpath>  
</java>`

# Tarefa jar

---

- Cria um arquivo **.jar** contendo os arquivos especificados

1. 

```
<jar jarfile="${dist}/lib/app.jar"
    basedir="${build}/classes" />
```
2. 

```
<jar jarfile="${dist}/lib/app.jar"
    basedir="${build}/classes"
    excludes="**/Test.class" />
```
3. 

```
<jar jarfile="${dist}/lib/app.jar">
    <fileset dir="${build}/classes"
        excludes="**/Test.class" />
    <fileset dir="${src}/resources" />
</jar>
```

# Tarefa <junit>

- Permite a execução de testes JUnit;
  - Opcionalmente, cria dados (xml) para a geração posterior de relatórios
- Exemplo:

```
<junit printsummary="yes" haltonfailure="true">  
  <classpath refid="test.classpath"/>  
  <formatter type="xml"/>  
  <test name="org...HtmlDocumentTest"/>  
  <batchtest todir="${test.data.dir}">  
    <fileset dir="${test.dir}"  
      includes="**/*Test.class" />  
  </batchtest>  
</junit>
```

Saída  
XML

Execução  
de vários  
test cases

# Tarefa <junitreport>

---

- Permite a geração de relatórios HTML de testes JUnit realizados anteriormente.
- Exemplo:

```
<junitreport todir="${test.data.dir}">
```

**Entrada  
XML**

```
<fileset dir="${test.data.dir}">  
  <include name="TEST-*.xml"/>  
</fileset>
```

```
<report format="frames"  
        todir="${test.reports.dir}"  />  
</junitreport>
```

# Exemplo de relatório <junitreport>

## [Home](#)

## Packages

[org.example.antbook.ant.lucene](#)  
[org.example.antbook.junit](#)

## Classes

[HtmlDocumentTest](#)  
[SimpleTest](#)

## Unit Test Results

Designed for use with [JUnit](#) and [Ant](#)

## Summary

Tests	Failures	Errors	Success rate	Time
2	1	0	50.00%	2.369

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

## Packages

Name	Tests	Errors	Failures	Time(s)
<a href="#">org.example.antbook.ant.lucene</a>	1	0	1	1.197
<a href="#">org.example.antbook.junit</a>	1	0	0	1.172



# Um Exemplo de Buildfile

---

```
<project name="MeuProjeto" default="dist"  
  basedir=".">
```

```
  <property name="src" value="."/>
```

```
  <property name="build" value="build"/>
```

```
  <property name="dist" value="dist"/>
```

```
  <target name="init">
```

```
    <tstamp/>
```

```
    <mkdir dir="${build}"/>
```

```
  </target>
```

```
  <target name="compile" depends="init">
```

```
    <javac srcdir="${src}" destdir="${build}"/>
```

```
  </target>
```

# Um Exemplo de Buildfile (continuação)

---

(...)

```
<target name="dist" depends="compile">
  <mkdir dir="${dist}/lib"/>
  <jar jarfile="${dist}/lib/
    MyProject-${DSTAMP}.jar" basedir="${build}"/>
</target>
```

```
<target name="clean">
  <delete dir="${build}"/>
  <delete dir="${dist}"/>
</target>
</project>
```

# Executando o Ant

---

- Sintaxe padrão:

```
ant [opções] [alvos]
```

- Opções

-help	-logfile <i>arquivo</i>
-projecthelp	-logger <i>nomeClasse</i>
-version	-listener <i>nomeClasse</i>
-quiet	-buildfile <i>arquivo</i>
-verbose	-find <i>arquivo</i>
-debug	-D <i>propriedade=valor</i>

# Exemplos

---

```
ant
```

Executa o Ant usando o *buildfile* **build.xml** no diretório atual, a partir do alvo padrão

```
ant -buildfile test.xml
```

Executa o Ant usando o arquivo **test.xml** no diretório atual, a partir do alvo padrão

# Exemplos (continuação)

---

```
ant -buildfile test.xml dist
```

Executa o **Ant** usando o arquivo **test.xml** no diretório atual, a partir do alvo **dist**

```
ant -buildfile test.xml -Dvar=classes dist
```

Executa o **Ant** usando o arquivo **test.xml** no diretório atual, a partir do alvo **dist**, setando o valor da propriedade **var** para **classes**.

# Exercício

---

- Escrever um script build.xml de uma aplicação exemplo que:
  1. Compile o código fonte java principal da aplicação
  2. Gere Javadoc do projeto
  3. Compile o código fonte java de teste da aplicação
  4. Execute o código de teste
  5. Empacote a aplicação em um arquivo jar



# Boas Práticas

---

- Projete seus scripts
  - Padronize e simplifique scripts (*Keep It Simple **Sir***)
- Aumentar produtividade e reuso
  - Dentro e entre projetos
    - `<macrodef>` permite que você defina "métodos privados" com parâmetros, chamados atributos
    - `<import>` pode ser usado para realizar "herança" entre scripts (definir um script genérico para todos projetos)
- Gerencie dependências com bibliotecas
  - Gerenciar de dependências: Ivy

# Exemplo de reuso com <macrodef>

---

```
<target name="compile">  
  <compilecode srcdir="${source.java.dir}" classpath="classpath.main"/>  
  <compilecode srcdir="${unit.test.source.dir}" classpath="classpath.test"  
    includeant="true" />  
</target>
```

```
<macrodef name="compilecode">  
  <attribute name="srcdir"/>  
  <attribute name="includeant" default="false"/>  
  <attribute name="classpath"/>  
  <sequential>  
    <javac srcdir="@{srcdir}" destdir="${compile.dir}"  
      classpathref="@{classpath}" includeAntRuntime="@{includeant}"  
      debug="${compile.debug}" debugLevel="${compile.debugLevel}"  
      deprecation="${compile.deprecation}" optimize="${compile.optimize}" />  
  </sequential>  
</macrodef>
```




# Exemplo de reuso com `<element>` e `<macrodef>`

Uso do  
Macrodef

```
<doTests fork="no">
  <whatToTest>
    <test fork="yes" haltonerror="false" haltonfailure="false"
      name="@{className}" todir="${junit.report.dir}"/>
    </whatToTest>
  </doTests>
```

Qualquer coisa pode  
ser inserida no  
macrodef utilizando  
`<element>`



Definição do  
Macrodef

```
<macrodef name="doTests">
  <attribute name="fork" default="no"/>
  <element name="whatToTest" optional="no"/>
  <sequential>
    <junit printsummary="on" fork="@{fork}" showoutput="true"
      haltonfailure="false" failureproperty="test.failed" errorproperty="test.failed">
      <sysproperty key="app.root.dir" value="${app.root.dir}"/>
      <sysproperty key="fromant" value="yep"/>
      <classpath refid="runtest.classpath"/>
      <formatter type="xml"/>
      <formatter type="brief" usefile="false"/>
      <jvmarg value="-Demma.coverage.out.file=${coverage.dir}/metadata/coverage.emma"/>
      <jvmarg value="-Demma.coverage.out.merge=true"/>
    <whatToTest/>
  </junit>
</sequential>
</macrodef>
```

# Exemplo de reuso com <import>

## generico-build.xml

```
<?xml version="1.0"?>
<project name="master" >
  <target name="clean" >
    <echo >Limpa Geral</echo>
  </target>
  <target name="compile" depends="init">
    <echo >Compila Geral</echo>
  </target>
</project>
```

## build.xml

```
<?xml version="1.0"?>
<project name="cr" basedir=".." >
  <import file="generico-build.xml"/>
  <target name="clean" >
    <echo> Limpa Especifico</echo>
  </target>
  <target name="deploy" depends="compile"/>
  <target name="init" depends="clean" />
</project>
```

## Saída

```
ant deploy
Buildfile: build.xml
clean:
  [echo] Limpa Específico
init:
compile:
  [echo] Compila Geral
deploy:
BUILD SUCCESSFUL
```

## Observações:

- Clean é sobreescrito em build.xml
- init não é definido em generico-build.xml
- deploy utiliza compile em generico-build.xml

**Demonstração**

# Exercício

---

- Refatore o build.xml da aplicação utilizando:
  1. A tarefa <macrodef> para promover reuso;
  2. Extrair tarefas comuns à vários projetos para um build “genérico”
  3. Utilizar a tarefa <include>



# Onde encontrar o Ant

---

- <http://ant.apache.org/>
  - Site oficial do projeto
  - **Documentação extensiva**
  - Diversas versões disponíveis
  - Pacotes com tarefas adicionais
- <http://ant-contrib.sourceforge.net/>
  - Projeto Sourceforge com vários ant tasks úteis
- Apresentação de boas práticas em scripts Ant
  - Writing Better Ant Scripts: Techniques, Patterns and Antipatterns, Douglas Bullard. [www.pjug.org/docs/ant.pdf](http://www.pjug.org/docs/ant.pdf)

# Instalando o ANT

---

- Download da versão mais atual do Ant
  - <http://ant.apache.org/bindownload.cgi>
- Baixar e descompactar o arquivo:
  - apache-ant-1.8.2-bin.zip
  - No diretório `${INSTALACAO}`
- Criar a variável `ANT_HOME` apontando para o diretório `${INSTALACAO}`.
- Adicionar o diretório `${ANT_HOME}/bin` à variável `PATH`.
- Verificando instalação:
  - `$>ant -version`
  - Apache Ant version 1.8.2 compiled on June 3 2011



# **Gerenciando Dependências com Ivy**

# O que é mais descritivo?

---

## Algo assim ...

```
<fileset dir="${global.lib.dir}">
  <include name="commons-beanutils.jar"/>
  <include name="commons-collections.jar"/>
  <include name="commons-digester.jar"/>
  <include name="commons-logging.jar"/>
  <include name="commons-validator.jar"/>
  <include name="commons-resources.jar"/>
  <include name="jakarta-oro.jar"/>
  <include name="struts.jar"/>
  <include name="struts-el.jar"/>
  <include name="commons-lang.jar"/>
  <include name="jstl.jar"/>
  <include name="standard.jar"/>
  <include name="commons-pool.jar"/>
  <include name="displaytag.jar"/>
</fileset>
```

# O que é mais descritivo? ...

---

## Ou assim ...

<dependencies>

```
<dependency org="org.hibernate" name="hibernate" rev="3.2.0.ga"
  conf="dist-ear,source,javadoc"/>
```

```
<dependency org="org.apache" name="log4j" rev="1.2.8" conf="dist-ear"/>
```

```
<dependency org="org.apache" name="struts" rev="1.3.8" conf="dist-ear"/>
```

```
<dependency org="org.apache" name="struts-el" rev="1.3.8" conf="dist-ear"/>
```

```
<dependency org="org.springframework" name="spring" rev="2.0" conf="dist-ear"/>
```

</dependencies>



# Por que um Gerenciador de Dependências?

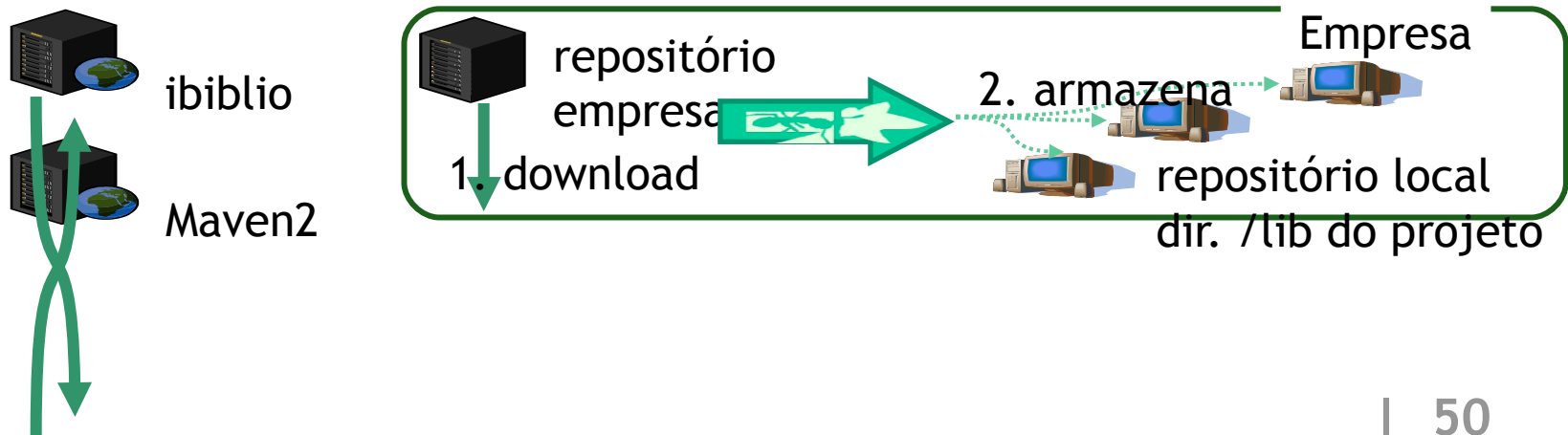
---

1. Sistemas dependem cada vez mais e mais de **componentes complexos**
2. Precisamos gerenciar **conflitos entre versões**
3. Garantir a **reproducibilidade** dos builds
4. Não reinvente a roda: reuse
5. Sistemas **modularizados e flexíveis** são fáceis de desenvolver e manter
6. Gerenciar Dependências: **Metodologia Ágil**

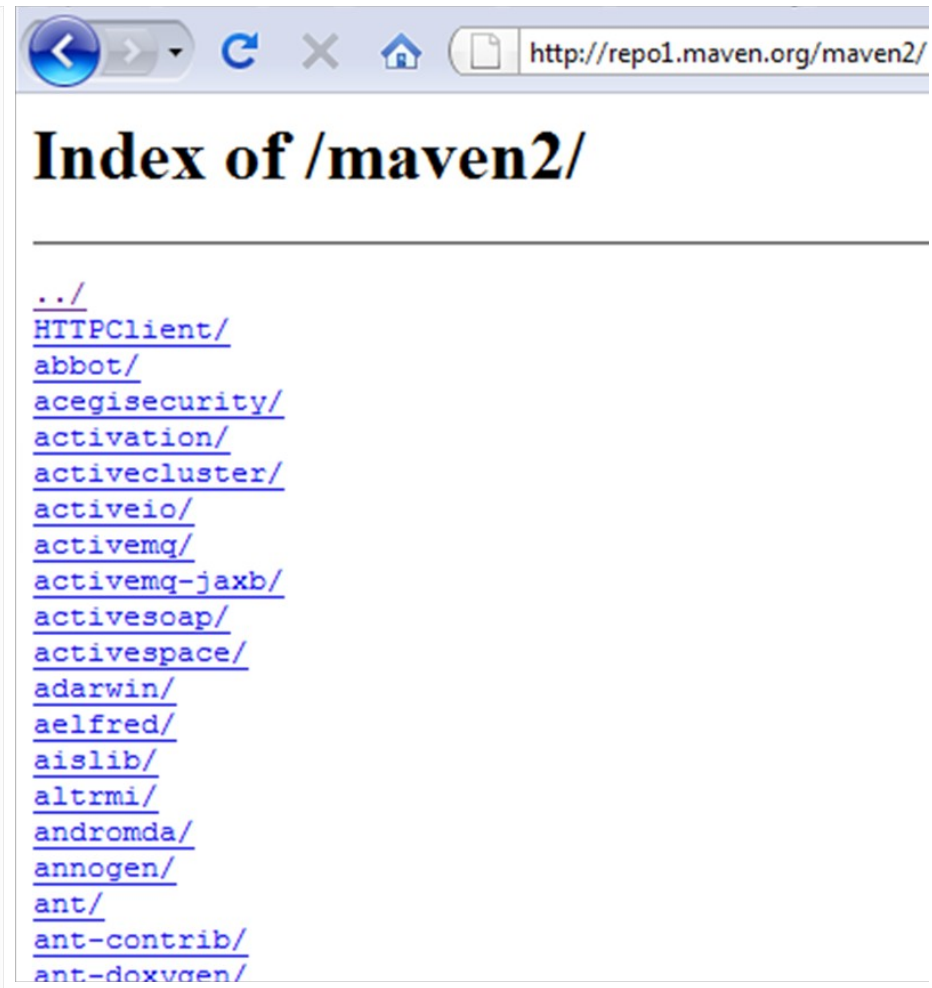
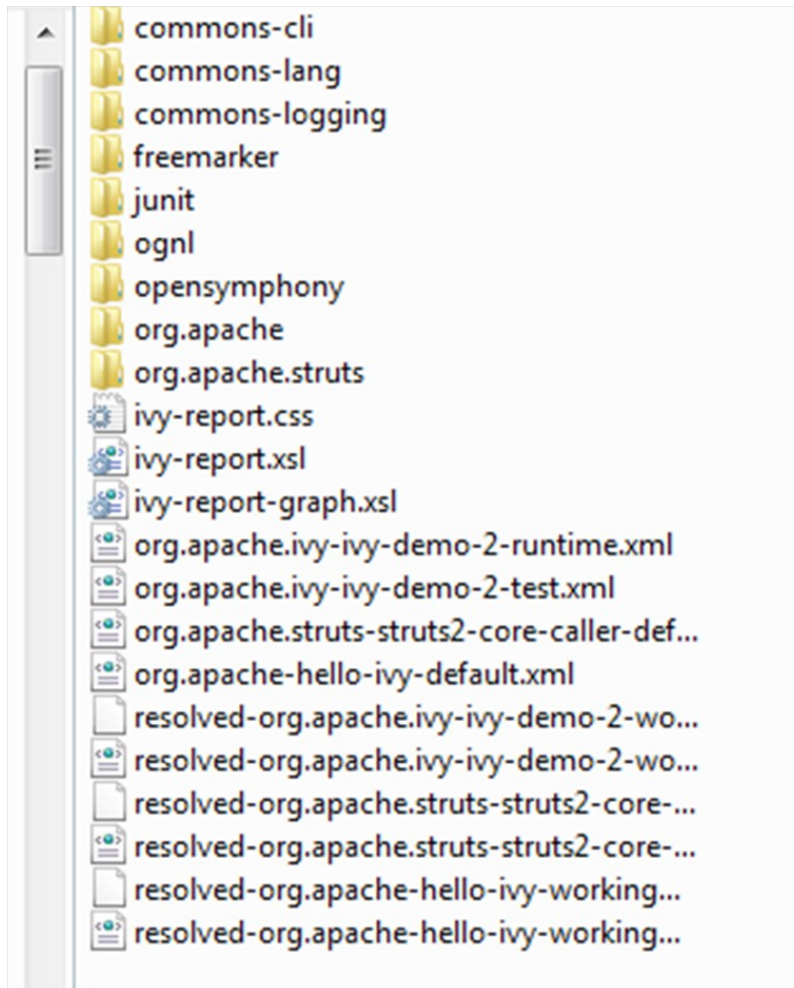


# Ivy + Ant

- Ivy projeto Apache | Site: <http://ant.apache.org/ivy/>
  - Gerência de dependências similar Maven2 (Ant + Ivy ~= Maven2)
  - Permite que controlemos quais versões estão disponíveis
    - Versões diferentes podem ser usadas em diferentes projetos
  - Suporta dependências transitivas [*dependências das dependências, ..., etc*]
    - Download das dependências do projeto e das dependências das dependências
  - Dependências são descritas em um arquivo específico
- Funcionamento:



# Exemplos de repositórios: Local e Remoto



# Instalando Ivy

- O Ivy normalmente é utilizado como um plugin ANT
  - Você colocar seu jar no diretório lib do ANT\_HOME
  - Ou, criar um target para fazer o download sob demanda

```
<target name="install-ivy" description="--> install ivy">
  <mkdir dir="${ivy.jar.dir}"/>
  <get src="${ivy.download.url}" usetimestamp="true"
    dest="${ivy.home}/${download.ivy.binaryfile}" />
  <untar src="${ivy.home}/${download.ivy.binaryfile}"
    dest="${ivy.home}" overwrite="false"
    compression="gzip" />
  <taskdef resource="org/apache/ivy/ant/antlib.xml"
    uri="antlib:org.apache.ivy.ant"
    classpath="${ivy.jar.file}"/>
</target>
```

# Dependência Inline

Para recuperar dependência com Ivy:

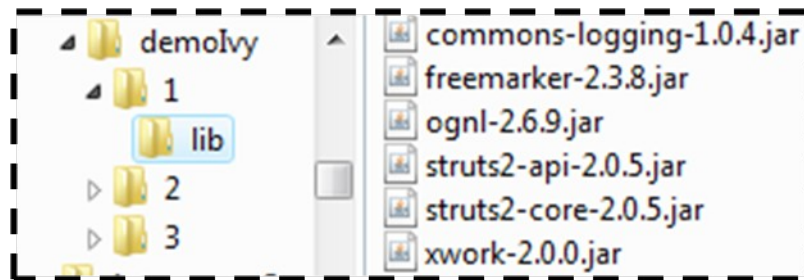
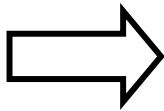
- basta adicionar uma tarefa Ivy (inline) no build



**build.xml**

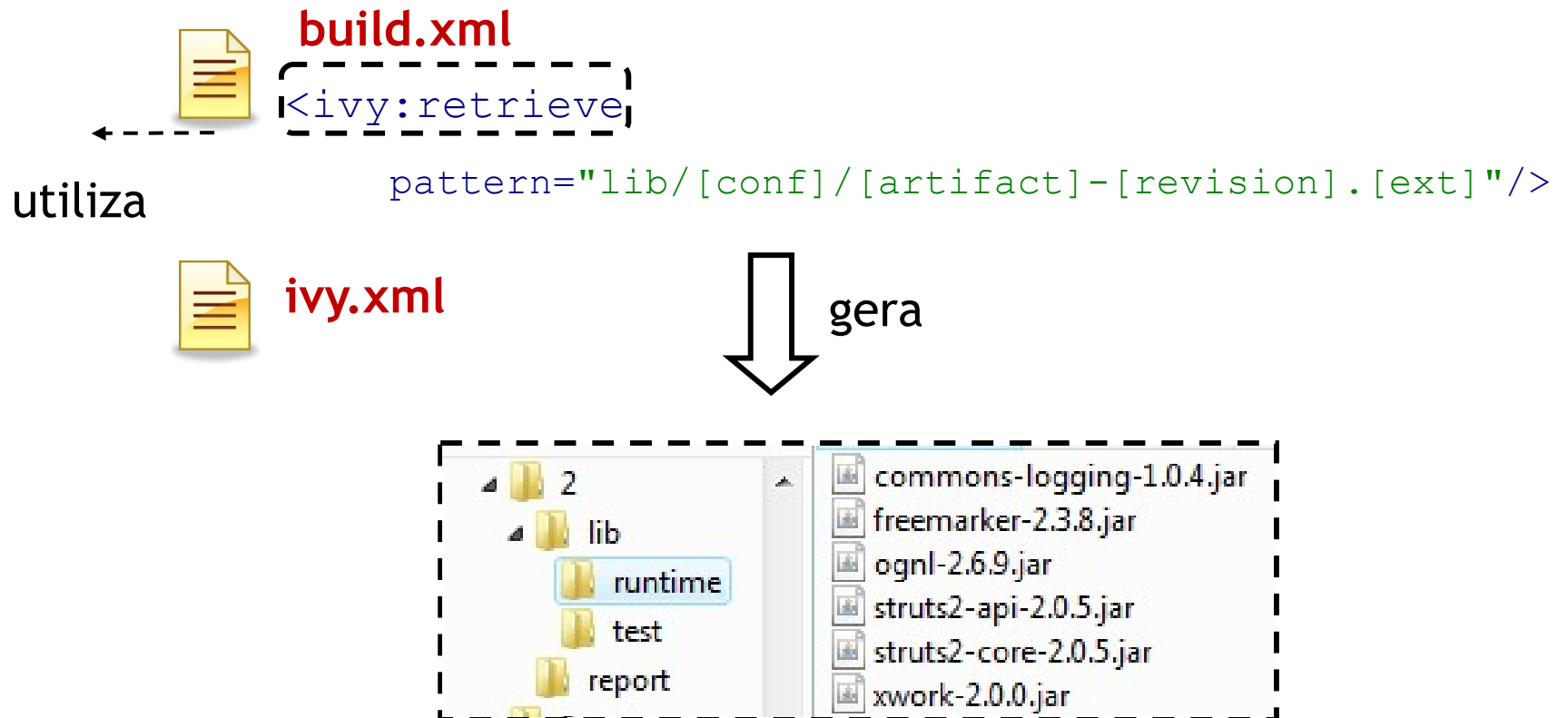
```
[<ivy:retrieve organisation="org.apache.struts"  
              module="struts2-core"  
              revision="2.0.5"  
              inline="true"/>]
```

gera



# Arquivos Ivy

- Ajudam a separar declarações de dependências dos scripts de build



# Arquivos Ivy

```
<ivy-module version="1.5">
  <info organisation="org.apache.ivy" module="ivy-demo-2" />
  <configurations>
    <conf name="runtime"/>
    <conf name="test"/>
  </configurations>
  <dependencies>
    <dependency org="org.apache.struts"
      name="struts2-core" rev="2.0.5"
      conf="runtime->default"/>
    <dependency org="junit" name="junit" rev="3.8.1"
      conf="test->*" />
  </dependencies>
</ivy-module>
```



## build.xml

```
<ivy:report todir="report"/>
```

ivy-demo-2 by org.apache.ivy





resolved on 2007-03-09 11:56:34





runtime

test

### Dependencies Stats

Modules	6
Revisions	6 (2 searched  , 0 downloaded  , 0 evicted  , 0 errors  )
Artifacts	6 (0 downloaded, 0 failed)
Artifacts size	3589 kB (0 kB downloaded, 3589 kB in cache)

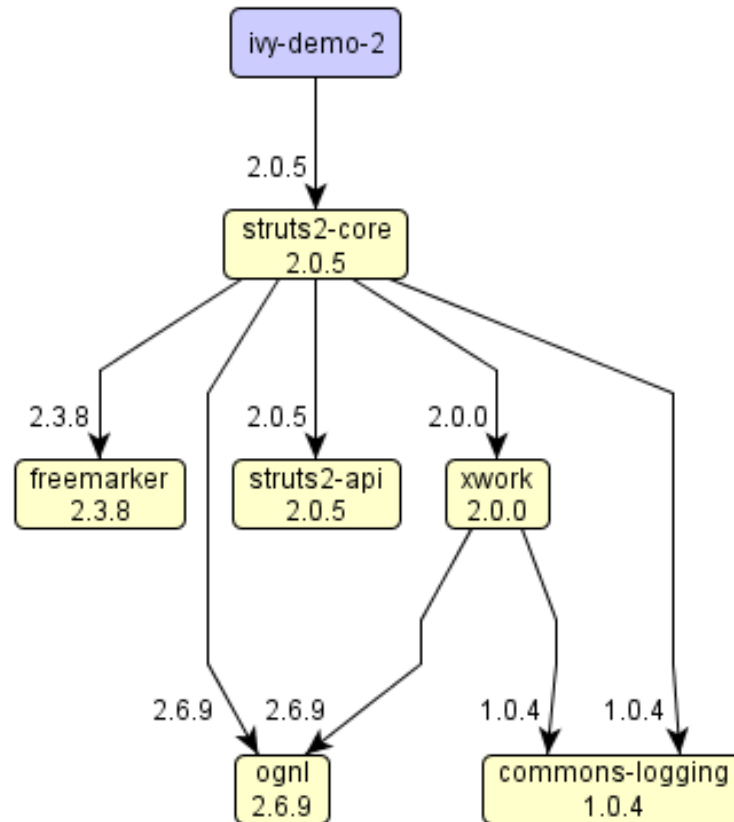
### Dependencies Overview

Module	Revision	Status	Resolver	Default	Licenses	Size	
struts2-core by org.apache.struts	2.0.5	integration	maven2-http	false		2154 kB	
--- freemarker by freemarker	2.3.8	integration	maven2-http	false		784 kB	
--- struts2-api by org.apache.struts	2.0.5	integration	maven2-http	false		12 kB	
--- xwork by opensymphony	2.0.0	integration	maven2-http	false		438 kB	
----- ognl by ognl	2.6.9	integration	maven2-http	false		164 kB	
----- commons-logging by commons-logging	1.0.4	integration	maven2-local	false		37 kB	
--- ognl by ognl	2.6.9	integration	maven2-http	false		164 kB	
--- commons-logging by commons-logging	1.0.4	integration	maven2-local	false		37 kB	



# Entendendo as dependências

---

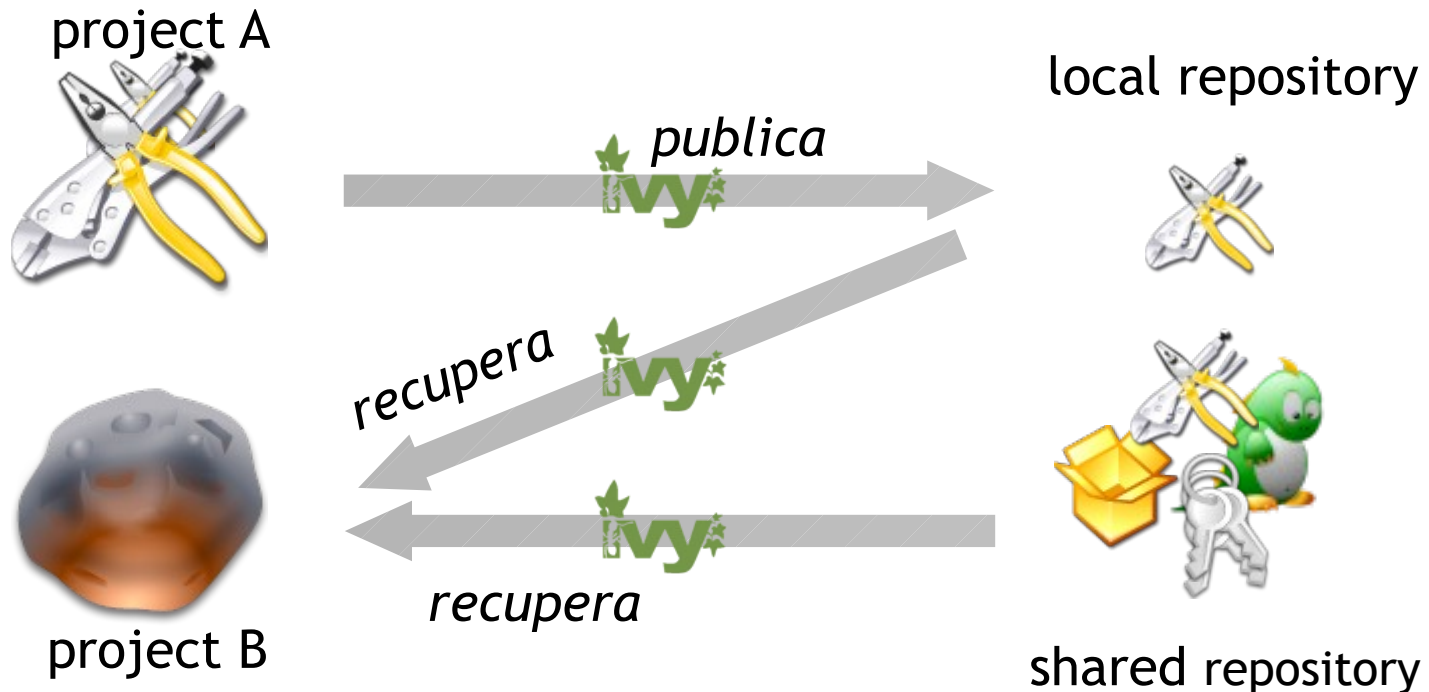


# Dependências entre projetos

---

*Ivy pode ser usado para resolver:*

- Dependências externas (com bibliotecas de terceiros)
- Dependências internas (entre projetos)



# Arquivo Ivy

---

```
<ivy-module version="1.5">
  <info organisation="org.apache.ivy"  module="ivy-demo-3-B" />
  <configurations>
    <conf name="runtime"/>
    <conf name="test"/>
  </configurations>
  <dependencies>
    <dependency org="org.apache.struts" conf="runtime->default"
      name="struts2-core" rev="2.0.5" />
    <dependency name="ivy-demo-3-A" rev="latest.integration" />
  </dependencies>
</ivy-module>
```

# Exercício

---

- Crie um arquivo ivy.xml para gerenciar as dependências da aplicação.



# Onde Você pode encontrar o Ivy

---

- <http://incubator.apache.org/ivy>
  - Site oficial do projeto
  - Documentação
- Plugins Eclipse
  - <http://ant.apache.org/ivy/ivyde/>