Lecture 15: [Construction II] Version Control and System Building

Adriana Sejfia

School of Informatics, University of Edinburgh

This Lecture: Topics Covered

Construction: version control and system building

- The problem of systems changing
- Software Configuration Management
- Version control overview
  - How file updates can be lost
  - Lock-Modify-Unlock model (RCS)
  - Copy-Modify-Merge model (CVS and SVN) and three-way merge
  - Distributed version control (Git, Mercurial, Bazaar)
  - More on branches
- Build tools: Make, Ant, Maven, Gradle

The Problem of Systems Changing

- Systems are constantly changing through development and use
  - Requirements change and systems evolve to match
  - Bugs found and fixed
  - New hardware and software environments are targeted

The Problem of Systems Changing

- Systems are constantly changing through development and use
  - Requirements change and systems evolve to match
  - Bugs found and fixed
  - New hardware and software environments are targeted
- Multiple versions might have to be maintained at each point in time

The Problem of Systems Changing

- Systems are constantly changing through development and use
  - Requirements change and systems evolve to match
  - Bugs found and fixed
  - New hardware and software environments are targeted
- Multiple versions might have to be maintained at each point in time
- Easy to lose track of which changes were realised in which version

The Problem of Systems Changing

- Systems are constantly changing through development and use
  - Requirements change and systems evolve to match
  - Bugs found and fixed
  - New hardware and software environments are targeted
- Multiple versions might have to be maintained at each point in time
- Easy to lose track of which changes were realised in which version
- Help is needed in managing versions and the processes that produce them

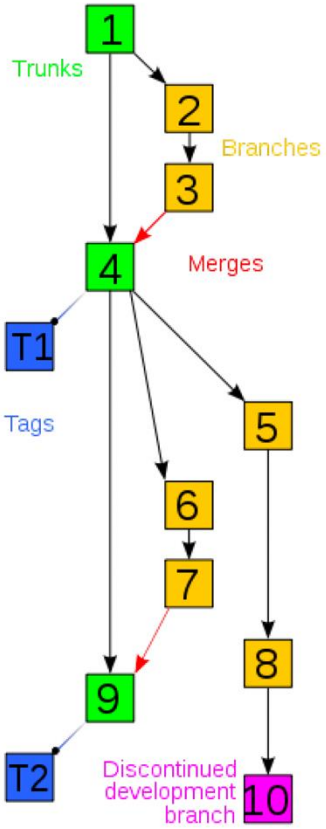
Software Configuration Management

CM provides help managing the evolution of complex systems. Provides activities such as:

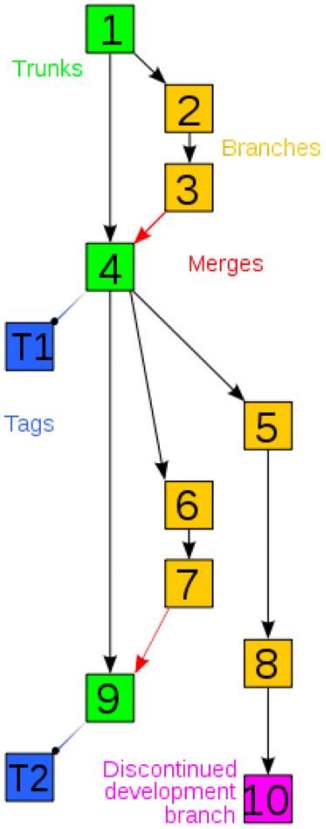
- Version control:
  - tracking multiple versions;
  - ensuring changes by multiple developers don't interfere
- System building:
  - assembling components, data and libraries;
  - compiling and linking to create executables;
  - an also be used to automatically package binary code and run tests
- Change management:
  - tracking change requests;
  - estimating difficulty and priority;
  - scheduling changes
- Release management:
  - preparing software for external release;
  - tracking released versions

Focus today: version control and system building

Version Control



Version Control



Important terminology:

- **Check in or commit:** push your changes to the central repo
- **Check out:** pull the changes from the central repo to your local repo

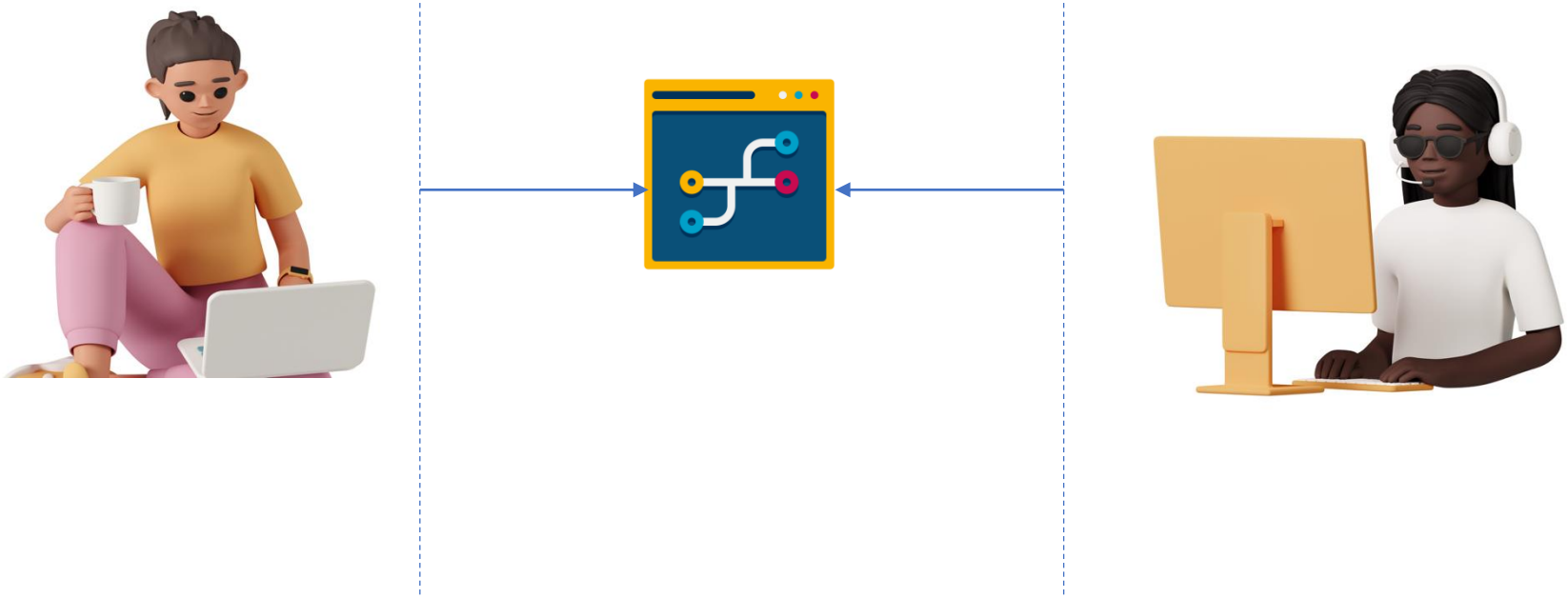
Version Control

The core of configuration management

- Keep copies of every version (every edit?) of files
- Provide change logs
- Manage the situation where several people want to edit the same file
- Provide diffs/deltas between versions

How do you manage your code?

When you work with others…



How do you manage your code?

How do you manage your code?

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomething(x);
6 }
7 ...
```

How do you manage your code?

10:00AM

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomethingElse(x);
6 }
7 ...
```

```
1 doSomethingElse(int x){
2   computeY(x);
3   computeZ(x);
4 }
```

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomething(x);
6 }
7 ...
```

How do you manage your code?

11:00AM

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomethingElse(x);
6 }
7 ...
```

```
1 doSomethingElse(int x){
2   computeY(x);
3   computeZ(x);
4 }
```

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomething(x);
6 }
7 ...
```

```
1 ...
2 int x;
3 if (conditionB){
4   x = 0;
5
6 }
7 ...
```

How do you manage your code?

4:00PM

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomethingElse(x);
6 }
7 ...
```

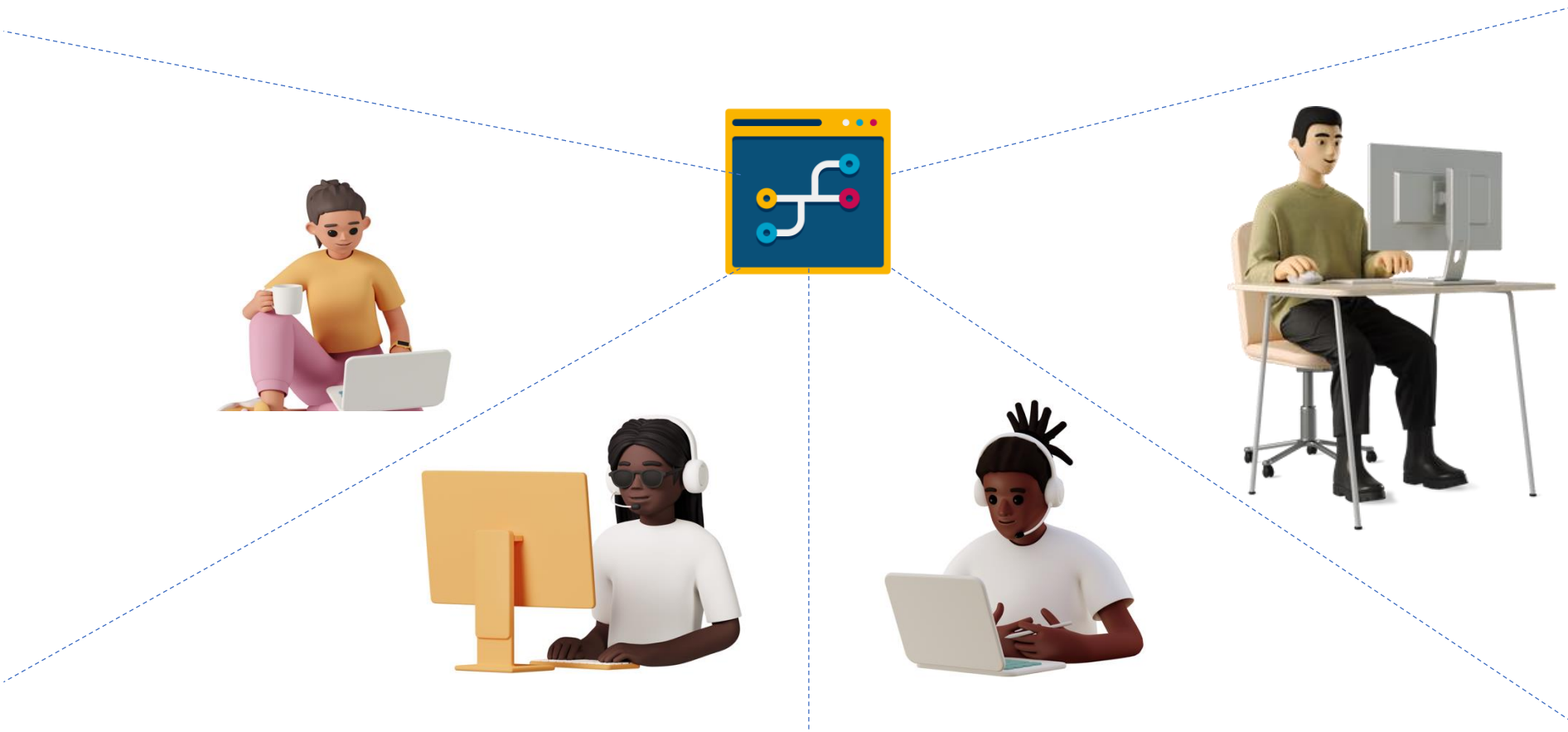
```
1 doSomethingElse(int x){
2   computeY(x);
3   computeZ(x);
4 }
```

```
1 ...
2 int x;
3 if (conditionB){
4   x = 0;
5
6 }
```

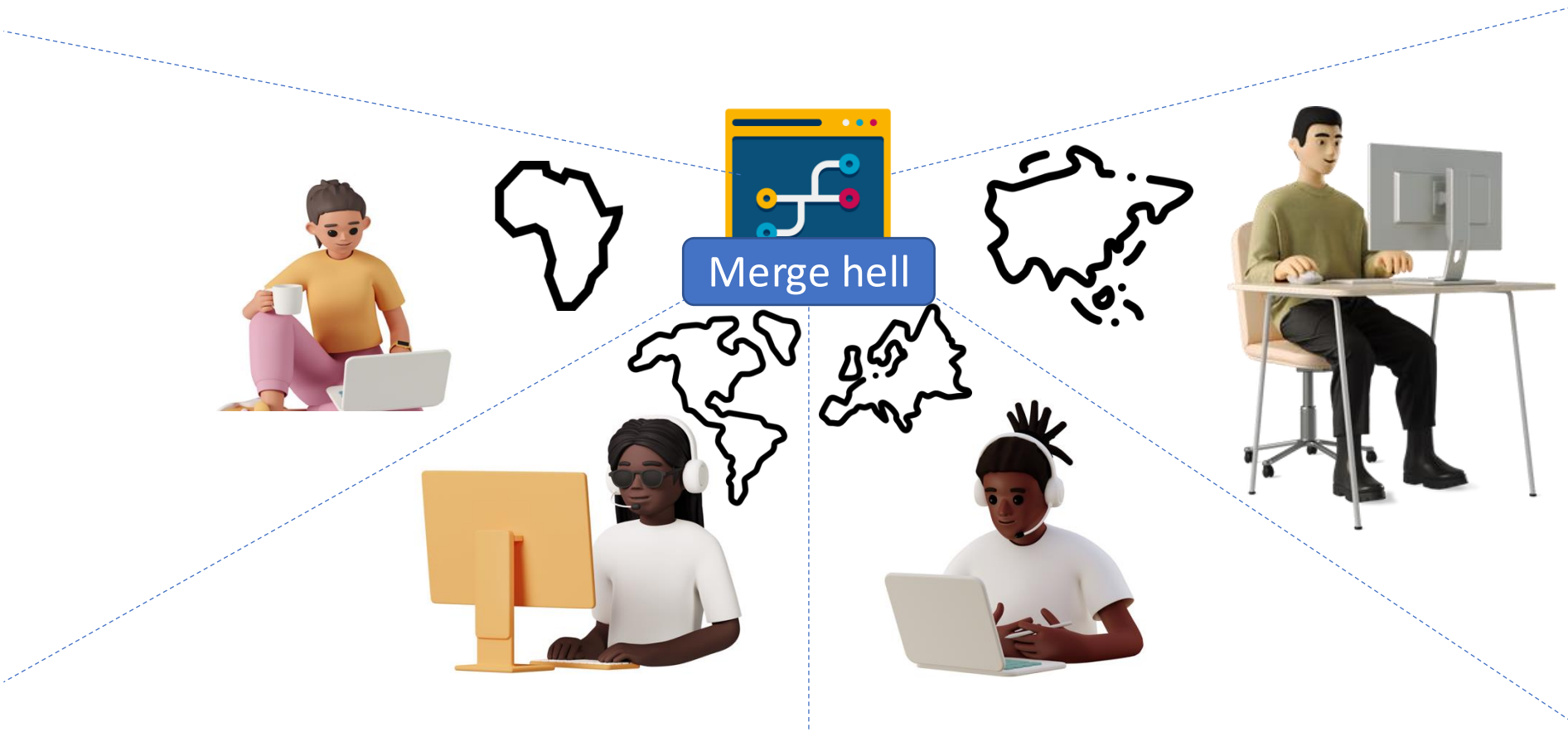
File will be overwritten

```
1 ...
2 int x;
3 if (conditionB){
4   x = 0;
5
6 }
7 ...
```

How is code managed in big companies?



How is code managed in big companies?



How File Updates Can Be Lost

- Two developers both read the same file from the repository
- Each makes their own edits independently (File' and File'')
- Developer 1 writes their version back first
- Developer 2 writes their version back next — overwriting Developer 1's changes
- Result: Developer 1's changes are silently lost
- Version control systems are designed to prevent this problem

Lock-Modify-Unlock Model (RCS)

- Editor checks out a file from the repository and locks it
  - Others can check out the file, but only for reading

Lock-Modify-Unlock Model (RCS)

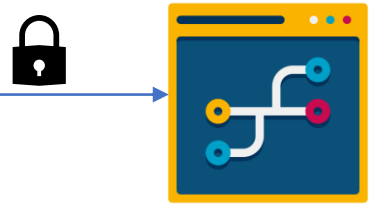
- Editor checks out a file from the repository and locks it
  - Others can check out the file, but only for reading
- Editor makes their changes

Lock-Modify-Unlock Model (RCS)

- Editor checks out a file from the repository and locks it
  - Others can check out the file, but only for reading
- Editor makes their changes
- Editor checks the modified file back in
  - Lock is released
  - Changes are now viewable by others, who can now make their own changes

How do you manage your code?

10:00AM



```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomethingElse(x);
6 }
7 ...
```

```
1 doSomethingElse(int x){
2   computeY(x);
3   computeZ(x);
4 }
```

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomething(x);
6 }
7 ...
```

Lock-Modify-Unlock Model (RCS)

- Editor checks out a file from the repository and locks it
  - Others can check out the file, but only for reading
- Editor makes their changes
- Editor checks the modified file back in
  - Lock is released
  - Changes are now viewable by others, who can now make their own changes
- Used by RCS: old Unix system, works on single files, best for small projects with one editor at a time
  - Keeps deltas, can restore, compare etc

CVS and SVN: Copy-Modify-Merge

- CVS is richer, based on RCS
- Subversion is similar but newer

CVS and SVN: Copy-Modify-Merge

- CVS is richer, based on RCS
- Subversion is similar but newer
- Handle entire directory hierarchies or projects with a single master repository

CVS and SVN: Copy-Modify-Merge

- CVS is richer, based on RCS
- Subversion is similar but newer
- Handle entire directory hierarchies or projects with a single master repository
- Designed for multiple developers working simultaneously

CVS and SVN: Copy-Modify-Merge

- CVS is richer, based on RCS
- Subversion is similar but newer
- Handle entire directory hierarchies or projects with a single master repository
- Designed for multiple developers working simultaneously
- Pattern of use:
  - Check out entire project or subdirectory — not individual files
  - Edit files locally
  - Update to get latest versions; system merges non-overlapping changes
  - User resolves overlapping changes — conflicts
  - Check in version with merges and resolved conflicts

CVS and SVN: Copy-Modify-Merge

- CVS is richer, based on RCS
- Subversion is similar but newer
- Handle entire directory hierarchies or projects with a single master repository
- Designed for multiple developers working simultaneously
- Pattern of use:
  - Check out entire project or subdirectory — not individual files
  - Edit files locally
  - Update to get latest versions; system merges non-overlapping changes
  - User resolves overlapping changes — conflicts
  - Check in version with merges and resolved conflicts
- Central repository may be on local filesystem or remote

How do you manage your code?

10:00AM

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomethingElse(x);
6 }
7 ...
```

```
1 doSomethingElse(int x){
2   computeY(x);
3   computeZ(x);
4 }
```

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomething(x);
6 }
7 ...
```

How do you manage your code?

11:00AM

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomethingElse(x);
6 }
7 ...
```

```
1 doSomethingElse(int x){
2   computeY(x);
3   computeZ(x);
4 }
```

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomething(x);
6 }
7 ...
```

```
1 ...
2 int x;
3 if (conditionB){
4   x = 0;
5
6 }
7 ...
```

How do you manage your code?

4:00 PM

```
1 ...
2 int x;
3 if (conditionA){
4   x = 0;
5   doSomethingElse(x);
6 }
7 ...
```

```
1 doSomethingElse(int x){
2   computeY(x);
3   computeZ(x);
4 }
```

```
1 ...
2 int x;
3 if (conditionB){
4   x = 0;
```

Update first

```
7 ...
```

```
1 ...
2 int x;
3 if (conditionB){
4   x = 0;
5
6 }
7 ...
```

How do you manage your code?

```
1 ...
2 int x;
3 if (conditionB){
4   x = 0;
5   doSomethingElse(x);
6 }
7 ...
```

```
1 ...
2 int x;
3 if (conditionB){
4   x = 0;
5
6 }
7 ...
```

```
1 ...
2 int x;
3 if (conditionB){
4   x = 0;
5
6 }
7 ...
```

```
1 doSomethingElse(int x){
2   computeY(x);
3   computeZ(x);
4 }
```

Three-Way Merge

1. Original file

(common ancestor):

Alpha,
Bravo,
Charlie

Three-Way Merge

1. Original file

(common ancestor):

Alpha,
Bravo,
Charlie

2. Tester 1 edits:

Alpha,
Foxtrot,
Charlie

Three-Way Merge

1. Original file

(common ancestor):

Alpha,
Bravo,
Charlie

2. Tester 1 edits:

Alpha,
Foxtrot,
Charlie

3. Tester 2 edits:

Delta,
Alpha,
Echo,
Charlie

Three-Way Merge

1. Original file (common ancestor):
   Alpha,
   Bravo,
   Charlie
2. Tester 1 edits:
   Alpha,
   Foxtrot,
   Charlie
3. Tester 2 edits:
   Delta,
   Alpha,
   Echo,
   Charlie
4. Tester 2 commits

Three-Way Merge

1. Original file (common ancestor):
   Alpha,
   Bravo,
   Charlie
2. Tester 1 edits:
   Alpha,
   Foxtrot,
   Charlie
3. Tester 2 edits:
   Delta,
   Alpha,
   Echo,
   Charlie
4. Tester 2 commits
5. Tester 1 commit fails

Three-Way Merge

1. Original file (common ancestor):
   Alpha,
   Bravo,
   Charlie
2. Tester 1 edits:
   Alpha,
   Foxtrot,
   Charlie
3. Tester 2 edits:
   Delta,
   Alpha,
   Echo,
   Charlie
4. Tester 2 commits
5. Tester 1 commit fails
6. Tester 1 updates and merges conflicts:
   Delta,
   Alpha,
   <<<<< .mine
   Foxtrot,
   ======
   Echo,
   >>>>>> .r4
   Charlie

Distributed Version Control (Git, Mercurial, Bazaar)

- All tools so far use a single central repository
  - Cannot check in unless you connect
  - Permission required

Distributed Version Control (Git, Mercurial, Bazaar)

- All tools so far use a single central repository
  - Cannot check in unless you connect
  - Permission required
- Distributed VCS allows many repositories of the same software to be managed and merged

Distributed Version Control (Git, Mercurial, Bazaar)

- All tools so far use a single central repository
  - Cannot check in unless you connect
  - Permission required
- Distributed VCS allows many repositories of the same software to be managed and merged
- Advantages:
  - Reduces dependence on a single physical node
  - Allows work including check-ins while disconnected
  - Much faster VC operations

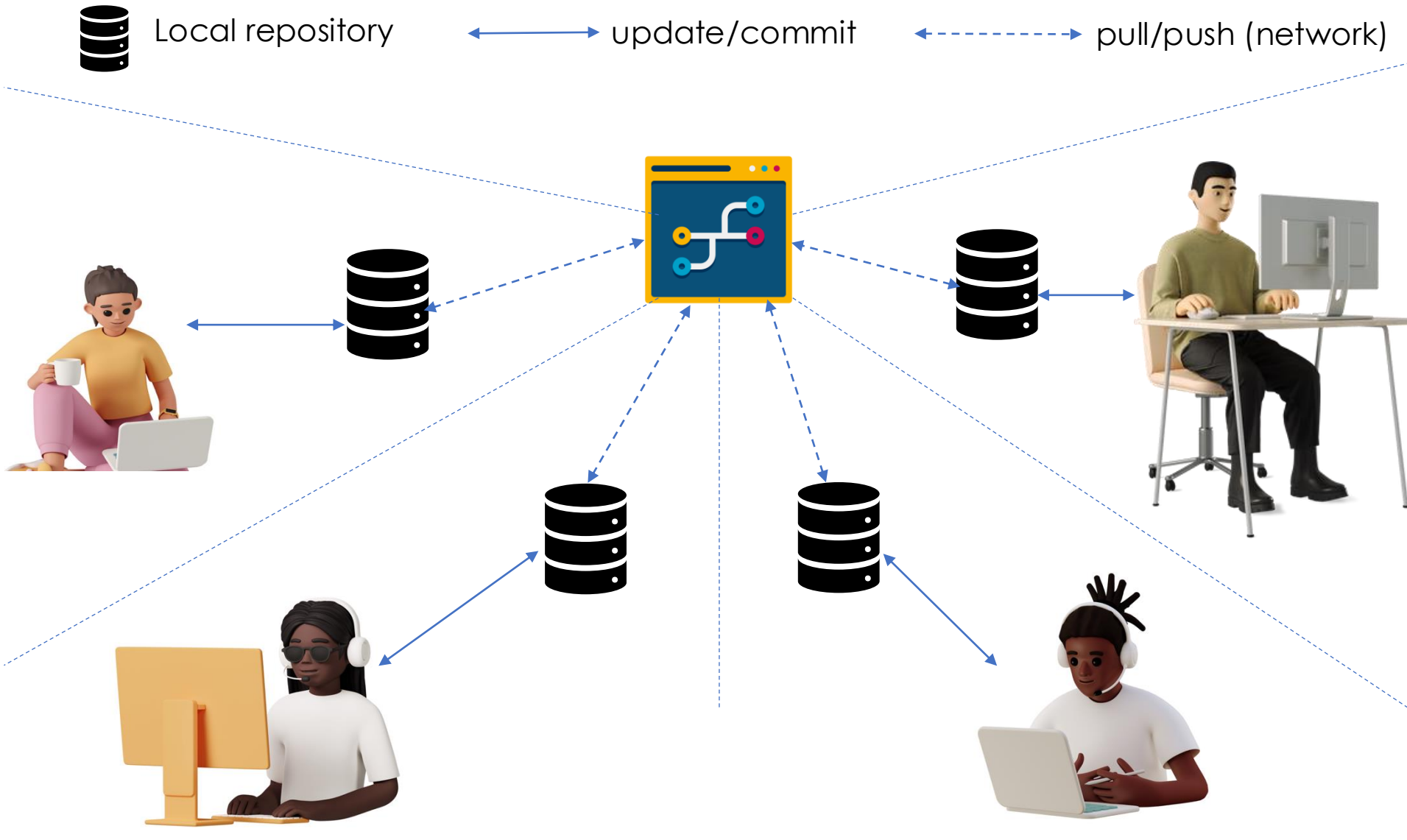
Distributed Version Control (Git, Mercurial, Bazaar)

- All tools so far use a single central repository
  - Cannot check in unless you connect
  - Permission required
- Distributed VCS allows many repositories of the same software to be managed and merged
- Advantages:
  - Reduces dependence on a single physical node
  - Allows work including check-ins while disconnected
  - Much faster VC operations
- Disadvantage: much more complicated and harder to understand

Distributed Version Control (Git, Mercurial, Bazaar)

- All tools so far use a single central repository
  - Cannot check in unless you connect
  - Permission required
- Distributed VCS allows many repositories of the same software to be managed and merged
- Advantages:
  - Reduces dependence on a single physical node
  - Allows work including check-ins while disconnected
  - Much faster VC operations
- Disadvantage: much more complicated and harder to understand
- Each developer has a full local repository; pushes and pulls to/from central or peer repos

How is code managed in big companies?



Branches

- Simplest VCS use gives a single linear sequence of versions

Branches

- Simplest VCS use gives a single linear sequence of versions
- Often need to maintain two versions simultaneously, e.g.:
  - The version customers are using — needs bug fixes
  - A new version under active development

Branches

- Simplest VCS use gives a single linear sequence of versions
- Often need to maintain two versions simultaneously, e.g.:
  - The version customers are using — needs bug fixes
  - A new version under active development
- Branching supports this — versions form a tree

Branches

- Simplest VCS use gives a single linear sequence of versions
- Often need to maintain two versions simultaneously, e.g.:
  - The version customers are using — needs bug fixes
  - A new version under active development
- Branching supports this — versions form a tree
- Merging branches requires a 3-way merge between branch ends and common ancestor

Branches

- Simplest VCS use gives a single linear sequence of versions
- Often need to maintain two versions simultaneously, e.g.:
  - The version customers are using — needs bug fixes
  - A new version under active development
- Branching supports this — versions form a tree
- Merging branches requires a 3-way merge between branch ends and common ancestor
- Git and Mercurial have strong merge support — developers use branches heavily
- SVN merge support has improved in recent versions

Build Tools: Make

- Given a large program in many files, how do you automatically recreate the executable when code changes?

Build Tools: Make

- Given a large program in many files, how do you automatically recreate the executable when code changes?
- On Unix the make command handles this, driven by a Makefile

Build Tools: Make

- Given a large program in many files, how do you automatically recreate the executable when code changes?
- On Unix the make command handles this, driven by a Makefile
- Used for C, C++ and other traditional languages — not language-dependent

Build Tools: Make

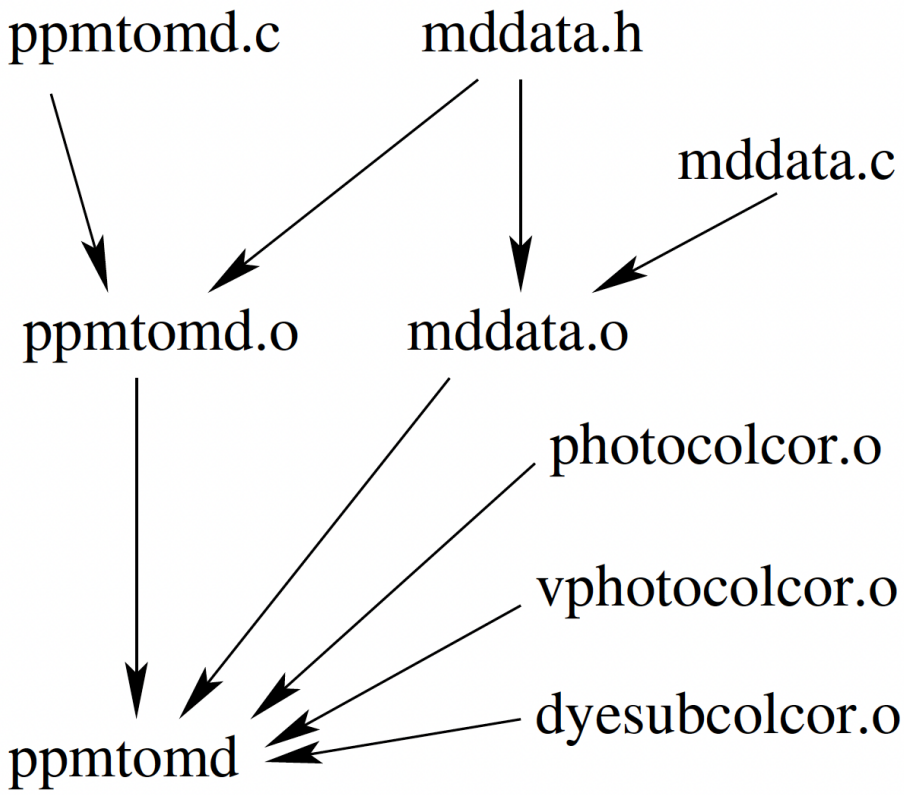
```
OBJS = ppmtomd.o mddata.o photocolcor.o vphotocolcor.o dyesubcolcor.o
ppmtomd: $(OBJS)
        $(CC) -o ppmtomd $(OBJS) $(LDLIBS) -lpnm -lppm -lpgm -lpbm -lm

ppmtomd.o: ppmtomd.c mddata.h
        $(CC) $(CDEBUGFLAGS) -W  -c ppmtomd.c

mddata.o: mddata.c mddata.h
```

- Makefile rule structure: target: dependencies, followed by commands
- Running make creates or rebuilds a target when any dependency is newer
- Before building, make recursively checks whether dependencies also need rebuilding

Make example



Ant: Build Tool for Java

- make can be used for Java, but Ant is a purpose-built Java build tool

Ant: Build Tool for Java

- make can be used for Java, but Ant is a purpose-built Java build tool
- Ant buildfiles (typically build.xml) are XML files

Ant: Build Tool for Java

- make can be used for Java, but Ant is a purpose-built Java build tool
- Ant buildfiles (typically build.xml) are XML files
- Specify the same kind of dependency and target information as make

Ant: Build Tool for Java

- make can be used for Java, but Ant is a purpose-built Java build tool
- Ant buildfiles (typically build.xml) are XML files
- Specify the same kind of dependency and target information as make
- Example targets: compile, jar, run, test

Ant: Build Tool for Java

- make can be used for Java, but Ant is a purpose-built Java build tool
- Ant buildfiles (typically build.xml) are XML files
- Specify the same kind of dependency and target information as make
- Example targets: compile, jar, run, test
- Paths and classpaths specified declaratively in XML

Ant: Build Tool for Java

- make can be used for Java, but Ant is a purpose-built Java build tool
- Ant buildfiles (typically build.xml) are XML files
- Specify the same kind of dependency and target information as make
- Example targets: compile, jar, run, test
- Paths and classpaths specified declaratively in XML
- Platform-independent — runs wherever Java runs

Ant example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name ="Dizzy" default = "run" basedir=".">
<description>
    This is an Ant build script for the Dizzy chemical simulator. [...]
</description>
<!-- Main directories -->
<property name = "source"      location = "${basedir}/src"/> [...]
<!--General classpath for compilation and execution-->
<path id="base.classpath">
  <pathelement location = "${lib}/SBWCore.jar"/> [...]
</path> [...]
<target name = "run" description = "runs Dizzy"
                     depends =" compile, jar">
  <java classname="org.systemsbiology.chem.app.MainApp" fork="true">
    <classpath refid="run.classpath" />
    <arg value="." />
  </java>
</target> [...]
</project>
```

Maven: Simplified Java Builds

- Designed to make building Java projects simpler and more uniform

Maven: Simplified Java Builds

- Designed to make building Java projects simpler and more uniform
- Provides default support for:
  - Compiling, testing (unit tests), packaging e.g. as a JAR
  - Integration testing, installing to local repo, deploying to release environments
  - Generating documentation
  - Deploy

Maven: Simplified Java Builds

- Designed to make building Java projects simpler and more uniform
- Provides default support for:
  - Compiling, testing (unit tests), packaging e.g. as a JAR
  - Integration testing, installing to local repo, deploying to release environments
  - Generating documentation
  - Deploy
- Buildfiles are XML files

Maven: Simplified Java Builds

- Designed to make building Java projects simpler and more uniform
- Provides default support for:
  - Compiling, testing (unit tests), packaging e.g. as a JAR
  - Integration testing, installing to local repo, deploying to release environments
  - Generating documentation
  - Deploy
- Buildfiles are XML files
- Convention over configuration — standard directory layouts reduce boilerplate

Maven: Simplified Java Builds

- Designed to make building Java projects simpler and more uniform
- Provides default support for:
  - Compiling, testing (unit tests), packaging e.g. as a JAR
  - Integration testing, installing to local repo, deploying to release environments
  - Generating documentation
  - Deploy
- Buildfiles are XML files
- Convention over configuration — standard directory layouts reduce boilerplate
- Dependency management via online repositories such as Maven Central

Gradle: Modern Build Tool

- Official build tool for Android; also used by LinkedIn, Netflix, Adobe, and many others

Gradle: Modern Build Tool

- Official build tool for Android; also used by LinkedIn, Netflix, Adobe, and many others
- Based on Groovy, a scripting language on the JVM — more concise than XML-based tools

Gradle: Modern Build Tool

- Official build tool for Android; also used by LinkedIn, Netflix, Adobe, and many others
- Based on Groovy, a scripting language on the JVM — more concise than XML-based tools
- Buildfiles can also be written in Kotlin

Gradle: Modern Build Tool

- Official build tool for Android; also used by LinkedIn, Netflix, Adobe, and many others
- Based on Groovy, a scripting language on the JVM — more concise than XML-based tools
- Buildfiles can also be written in Kotlin
- Highly configurable — handles complex or unusual build requirements

Gradle: Modern Build Tool

- Official build tool for Android; also used by LinkedIn, Netflix, Adobe, and many others
- Based on Groovy, a scripting language on the JVM — more concise than XML-based tools
- Buildfiles can also be written in Kotlin
- Highly configurable — handles complex or unusual build requirements
- More performant than Maven:
  - Incremental builds — only runs what is necessary
  - Build cache reduces redundant work
  - Gradle Daemon keeps build information in memory

Gradle: Modern Build Tool

- Official build tool for Android; also used by LinkedIn, Netflix, Adobe, and many others
- Based on Groovy, a scripting language on the JVM — more concise than XML-based tools
- Buildfiles can also be written in Kotlin
- Highly configurable — handles complex or unusual build requirements
- More performant than Maven:
  - Incremental builds — only runs what is necessary
  - Build cache reduces redundant work
  - Gradle Daemon keeps build information in memory
- Provides an interactive web-based UI for debugging and optimising builds

Reading

- On version control:
  - Essential: Ch 1- Version Control Basics section of the SVN book
  - (search for 'Version Control Basics' in table of contents)
  - Essential: Git & GitHub tutorial
  - Essential: A tutorial written by a former SEPP student

- On build tools:
  - Essential: Introduction to build tools
  - Recommended: Gradle tutorial
  - Recommended: Gradle Building Java Applications Sample
  - Recommended: Maven Tutorial
  - See other optional resources under 'Resources on other tools'