协同开发工具

曹东刚

caodg@sei.pku.edu.cn

Linux 程序设计环境

http://c.pku.edu.cn/

内容提要

1 patch

2 版本控制

3 问题跟踪系统

4 Daily Build

diff 和 patch

diff 和 patch 是 Unix 的源码补丁工具

- diff 以“行”为单位比较两个文本文件（也可以是目录比较），并将不同之处以某种格式输出到标准输出上；
- patch 可以读入这种输出，并按照一定指令使源文件（目录）按照目标文件（目录）更新。

diff

diff 用于生成源码补丁，其命令格式为:

```
diff [命令行选项] 原始文件 新文件
常用命令行选项如下:
  -r 递归处理目录
  -u 输出统一格式 (unified format)
  -N patch 里包含新文件
  -a patch 里可以包含二进制文件
```

diff的缺省输出在 stdout 上，实际可能需要把它重定向到一个文件

diff 的输出格式

diff 的输出有“命令格式”和“上下文格式”几种，现在大都使用上下文格式

- 命令模式分为两种:
  - ed 命令格式 (缺省或 –e 选项)
  - RCS（Revision Control System，版本控制系统）命令格式 (–n 选项)
- 上下文模式也按格式分为两种
  - 老版 (–c 选项，也称拷贝上下文)
  - 新版 (–u 选项，也称统一上下文)

命令格式

缺省情况下diff输出 ed 命令格式，特点是简洁，除了要删除/插入的行外没有冗余信息。

输出结果可以直接作为 ed 的命令控制文件

```
[caodg@debian]$ diff a.txt b.txt
1a2
> here we insert a new line
3d3
< why not this third line?
```

上下文格式

上下文格式保存了源文件上下文（缺省是上下各三行），同时在输出开头用---和+++ 标示出原始文件和当前文件，方便阅读

```
[caodg@debian]$ diff -u a.txt b.txt
--- a.txt Thu Apr 6 15:58:34 2000
+++ b.txt Thu Apr 6 15:57:53 2000
@@ -1,3 +1,3 @@
This is line one
+here we insert a new line
and this is line two
-why not this third line?
```

patch

- patch命令把diff 生成的补丁应用到现有代码上
- patch本身支持多种diff输出文件格式
- 对目标文件应用两次patch，则还原为原来的文件
- 如果 patch 成功，缺省不建备份文件
- 如果 patch 失败，patch会把成功的 patch 行给 patch 上同时（无条件）生成备份文件和一个.rej 文件。

patch常用选项：

```
patch -p[patch level] < patchfile
```

patch level

diff生成的 patch 文件里保存了目录路径, 通常如果当前目录树根目录和 patch 文件中的路径不一定匹配, 此时直接应用 patch 会失败.

```
diff -Nur p1/hello2.c p2/hello2.c
--- p1/hello2.c 2006-05-07 00:03:38.000000000 +0800
+++ p2/hello2.c 1970-01-01 08:00:00.000000000 +0800
```

patch level 就是为解决该问题而设: patch 会把目标路径名去掉开头 patch level 个目录 (由/分开的部分)

```
cd p1
patch -p1 < ../patch.p
```

应用 diff 和 patch

假定 program-1.0 目录中为老版，现开发完成的新版位于 program-2.0 目录中，要将程序的 1.0 版本升级为 2.0 版本:

- 将两个目录置于同一父目录下，在父目录生成 patch:

```
diff -Nur program-1.0 program-2.0 > program-2.0.patch
```

- 其他开发者拿到 patch 后在自己的 program-1.0 目录执行:

```
patch -p1 < program-2.0.patch
```

- 如此即完成了从 1.0 到 2.0 的升级

内容提要

1 patch

2 版本控制

3 问题跟踪系统

4 Daily Build

什么是版本控制

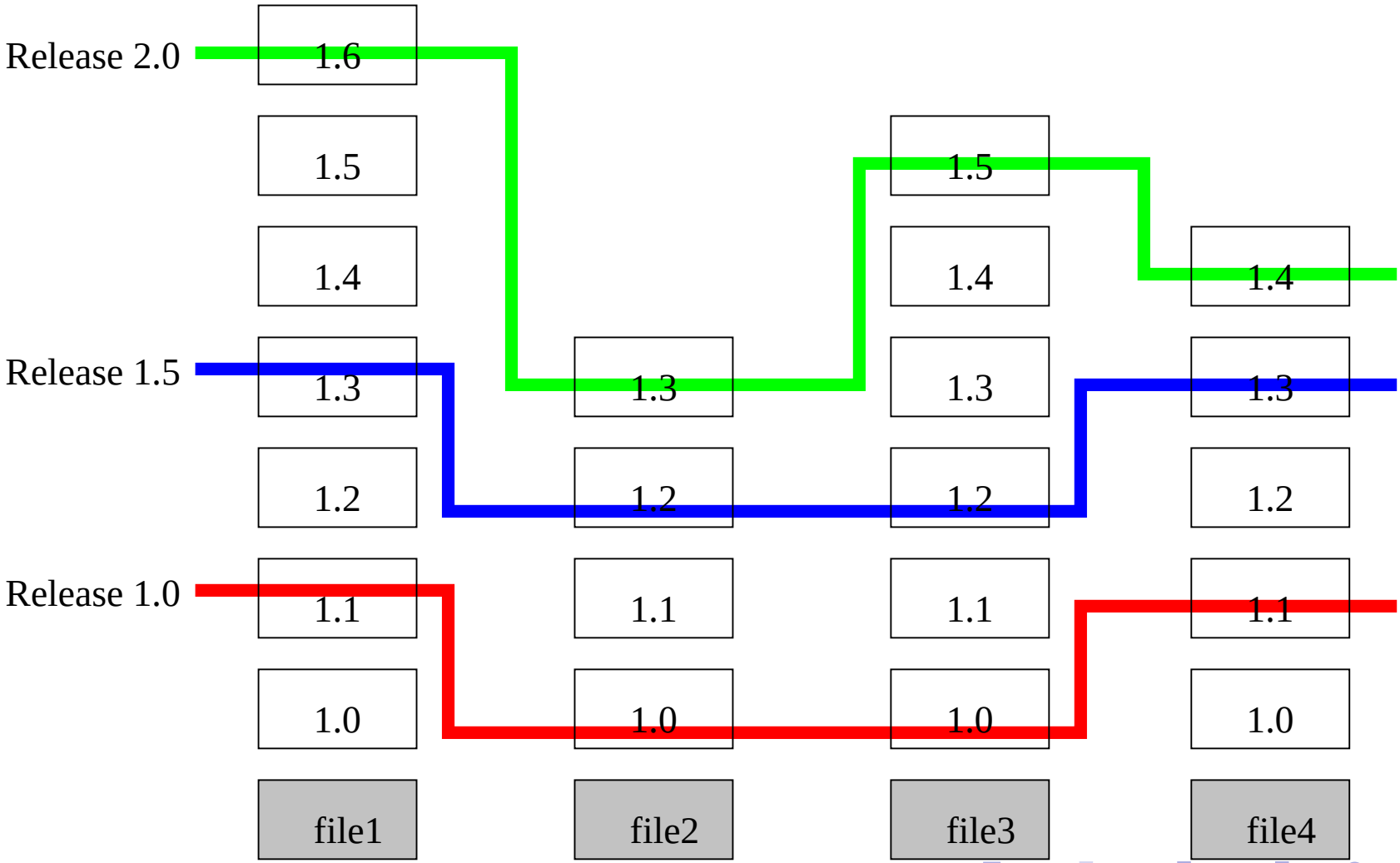
版本控制 (Version control, 又称 Revision control) 是一种软件工程活动, 对软件配置项 (Software configuration item: 程序, 文档及数据等) 的变动进行管理。

- 最简单的版本控制就是手工保留软件不同版本的数份 copy, 并且适当编号
  - 效率低, 易错, 不适用于中大型项目, 不适用于多人协作开发

为什么需要版本控制

- 为整个项目提供 Undo 机制
- 帮助多个人同时修改同一个代码段
- 能记录所有的变动
- 支持多个发布版本，而不影响开发主线
- 可查看任何日期的项目状态

版本变动


Release 2.0
Release 1.5
Release 1.0
1.6
1.5
1.4
1.3
1.2
1.1
1.0
1.5
1.4
1.3
1.2
1.1
1.0
1.4
1.3
1.2
1.1
1.0
file1
file2
file3
file4

版本控制系统

版本控制系统: 一组计算机程序, 用于自动维护软件配置项的所有版本, 记录一个软件配置项的版本变更历史, 即已做变更的内容、变更日期、变更人的姓名以及变更的原因等

- 源码控制系统 (Source Code Control System, SCCS)
- 修订控制系统 (Revision Control System, RCS)
- 并发版本系统 (Concurrent Version System, CVS)
- CVS 替代系统 (Subversion, SVN)
- 分布式版本控制系统 (GIT)

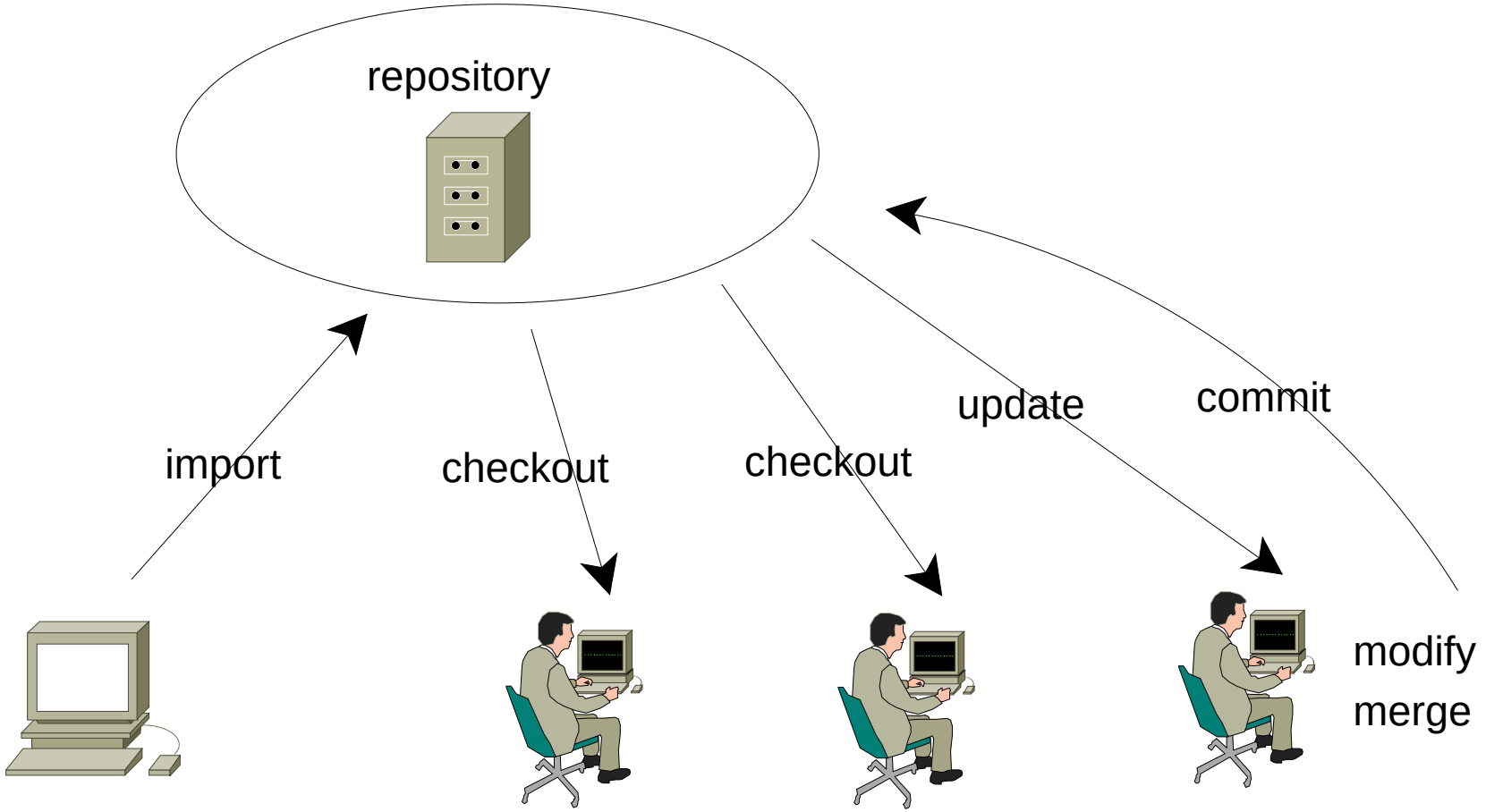
版本控制系统主要功能

- 保存任意一个软件配置项的版本变更历史
- 当两个用户同时修改一个文件时，尽可能地自动合并修改；在不能合并时，给出提示
- 比较不同版本之间的差异
- 可以回退到所保存的源代码的任意版本
- 可以创建代码分支，便于软件并行开发和维护；代码分支可以合并
- 对软件配置项进行标记，方便日后审查
- 阻止未经授权的修改和查阅

版本控制系统工作原理

- 大部分的版本控制软件采用增量存储方式: 只保留档案各版本之间的差异
- 传统上版本控制系统都是采用集中模式: 所有版本控制的工作在一个服务器进行, 程序员从服务器存储库 (repository) 签出 (checkout) 代码副本到本地工作空间, 在本地修改后提交 (commit) 到服务器
- 2000 年后, TeamWare, BitKeeper, 和 GNU 开始用分布式系统: 在分布式系统中开发者直接在各自的本地存储库工作, 各个存储库之间可以分享变更

集中式版本控制系统


repository
import
checkout
checkout
update
commit
modify
merge

什么应该由版本控制系统管理

- 源程序
- Makefile 或 build.xml
- 元数据
- 用于打包、测试、发布等的脚本文件

判断原则：如果不维护文件 x 的状态，项目就不能编译、发布，那么 x 应该放入版本控制系统

关键术语 –1

- 存储库 (repository): 保存所有软件配置项的完整修订历史的共享数据库, 位于服务器侧
- 工作空间 (workspace): 要在本地编辑的文件副本. 在工作空间中编辑文件后要将更改返回存储库, 以便其他人可以看到这些更改

关键术语 –2

- 主干 (trunk): 当前的主版本
- 标记 (tag): 用于标识文件集合的编号方案, 可在某个时间点标记并命名这些文件集
- 分支 (branch): 和主干并行的版本
- 修订版本号 (revision number): 版本控制系统标记各个文件更新的编号. 每次编辑文件并将它提交回存储库时, 该文件的修订版本号将会自动增加

关键术语 –3

- 导入 (import): 将某一个项目 (项目根目录) 导入版本控制系统, 对之进行版本控制
- 签出 (checkout): 从存储库中将文件的最新修订版本复制到工作空间. 签出目录时，将签出该目录下的所有文件和子目录。

关键术语 –4

- 提交 (commit): 将更改后的工作副本返回存储库
- 更新 (update): 使工作副本与存储库同步。用户进行修改前, 应进行更新
- 冲突 (conflict): 当两名开发人员对同一文件的工作副本进行更改并提交时, 他们的工作可能会发生冲突
- 合并 (merge): 将对相同文件的不同工作副本进行的多个更改合并到存储库中

RCS

RCS 是目前 Unix 世界最广泛应用的版本控制系统, 非常适合一个人或小型团队开发的项目

- 在当前目录下创建 RCS 目录: `mkdir RCS`
- 将文件导入 RCS: `ci filename`
- 从 RCS 导出文件: `co filename`
- 将修改后的文件保存回 RCS: `ci filename`
- 查看 log: `rlog filename`

CVS

- CVS 于 1990 年代初期开发, 在开源社区得到广泛应用
- CVS 最初架构于 RCS 系统之上, 作为 RCS 的前端. 目前 CVS 完全和 RCS 独立
- CVS 适用于大中型项目的分布式开发

CVS 的锁定机制

- CVS 在文件被签出 (checkout) 后，并不对文件进行锁定；而是在文件被提交 (commit) 时检查冲突，并要求人工干预
  - 发生冲突的几率实际上非常小
  - 避免了“单人失效问题”(single person point of failure)

CVS 基本操作 –1

- 客户端设置环境变量

  ```
  CVSROOT=/var/cvsroot
  ```

  或者

  ```
  CVSROOT=:pserver:username@162.105.81.88:/var/cvsroot
  ```

- 登录 CVS 服务器

  ```
  cvs login
  ```

- 在 CVS 服务器上创建一个项目 (把当前目录导入服务器)

  ```
  cd $HOME/project_dir
  cvs import project_dir V1_0 R1_0
  ```

CVS 基本操作 –2

- 签出工作副本到本地

  `cvs checkout project_dir`

  此时可以对该副本进行修改
- 提交修改: 对工作副本做的任意改动 (增删改除) 都要提交后才会在服务器生效

  `cvs commit`
- 增加一个文件

  `cvs add [-kb] filename`

  `cvs commit`

  增加一个二进制文件要用`-kb` 选项.

CVS 基本操作 –3

- 同步本地工作副本和服务器的内容

  `cvs update`

  在每天工作前和工作之后 commit 之前都应当 update，以保证本地代码总是最新的，且和服务器的代码无冲突。

- 从项目中删除文件. 删除时, 应当先将某个源文件物理删除后, 再使用 remove 命令

  ```
  rm file_name
  cvs remove file_name
  cvs commit
  ```

CVS 基本操作 –4

- 查看修改历史

  ```
  cvs log file_name
  ```

- 查看文件不同版本的区别

  ```
  cvs diff -r1.3 -r1.5 file_name
  ```

  查看 1.3 版本与 1.5 版本的区别

  ```
  cvs diff file_name
  ```

  查看本地和库中最新版本文件的区别

- 标记版本号

  ```
  cvs tag release_version
  ```

CVS 缺点

- 不支持对目录的版本管理
- 不支持文件改名
- 对二进制文件的支持不够
- 分支功能难以使用
- 标记功能效率低下

Subversion

继承了 CVS 的优点，克服了其缺点：

- CVS 的大多数特征
- 目录，改名，文件元信息的版本管理
- 原子提交
- 分支、标记、合并操作非常容易
- Apache 通过 WebDAV/DeltaV 协议直接支持

subversion 基本用法

建立代码库	`svnadmin create /path/to/repos`
导入数据	`svn import`
签出数据	`svn checkout`
提交更新	`svn commit filename`
添加文件	`svn add`
删除文件	`svn delele`
复制文件	`svn copy`
移动文件	`svn move`
查询状态	`svn status`
检查不同	`svn diff`
同步工作目录	`svn update`
合并代码	`svn merge; svn resolve`

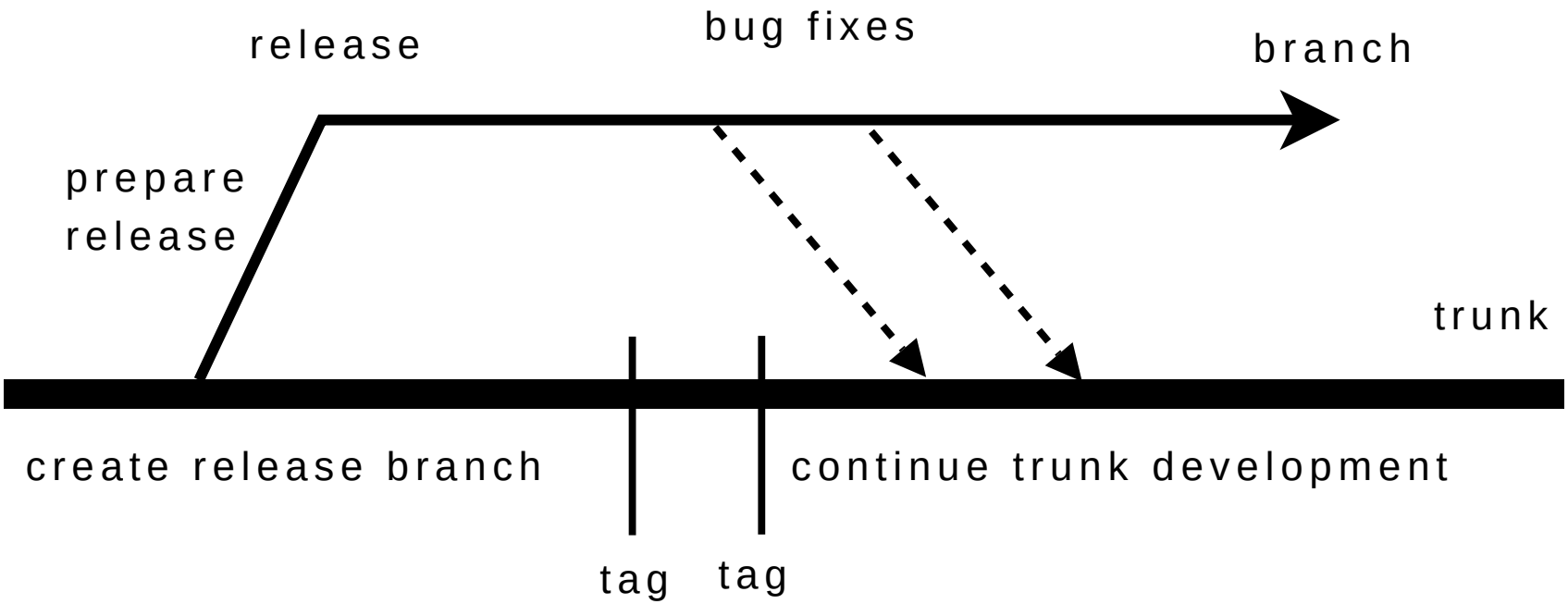
代码库的组织

假定 repos 是 SVN 的代码库, pkuas 是 repos 下面的一个项目, pkuas 的根目录组织:

- trunk
- branches
  - RB-1.0
  - BUG-3051
  - TRY-new-xml
- tags
  - REL-1.0

branch 和 tag

- branch 是独立的分支, 可进行 bug 修订等
- tag 只是某个特殊时间点项目的快照, tag 通常只读



创建和签出 branch

创建

```
work> svn copy -m "Creating release branch for 1.0" \
svn://olio/sesame/trunk \
svn://olio/sesame/branches/RB-1.0
Committed revision 33.
```

签出

```
work> svn co svn://olio/sesame/branches/RB-1.0 rb1.0
A rb1.0/Month.txt
A rb1.0/common
A rb1.0/common/Log.java
Checked out revision 33.
```

在版本间切换

```
work> cd sesame
sesame> svn switch svn://olio/sesame/branches/RB-1.0
U common/Clock.java
U contacts/Contacts.java
Updated to revision 36.

sesame> svn switch svn://olio/sesame/trunk
U common/Clock.java
U contacts/Contacts.java
Updated to revision 36.
```

git

分布式的版本管理系统，具有许多 subversion 不具备的优点

- 版本库是分布式的
  - 每个人都可以拥有一个自己的版本库
  - 提取操作实际上是一次对代码仓库的完整备份
  - commit 永远都会成功
  - 可以离线提交
  - 分支管理极其简单，不同用户彼此不干涉
- 版本库之间通过 pull 和 push 同步
- 速度快
- 可模拟 subversion

内容提要

1 patch

2 版本控制

3 问题跟踪系统

4 Daily Build

Issue Tracking System

问题跟踪系统: ITS

是专门用于记录、跟踪和管理各种问题的软件.

一贯地使用问题跟踪系统是优秀软件团队的标志特征之一

问题: Issue

对系统做出改进所需的一个工作单元. 典型的问题有: 缺陷 (Bug), 任务 (Task), 请求 (Request Feature), 等.

任务单: Ticket

对于问题的技术性描述单位.

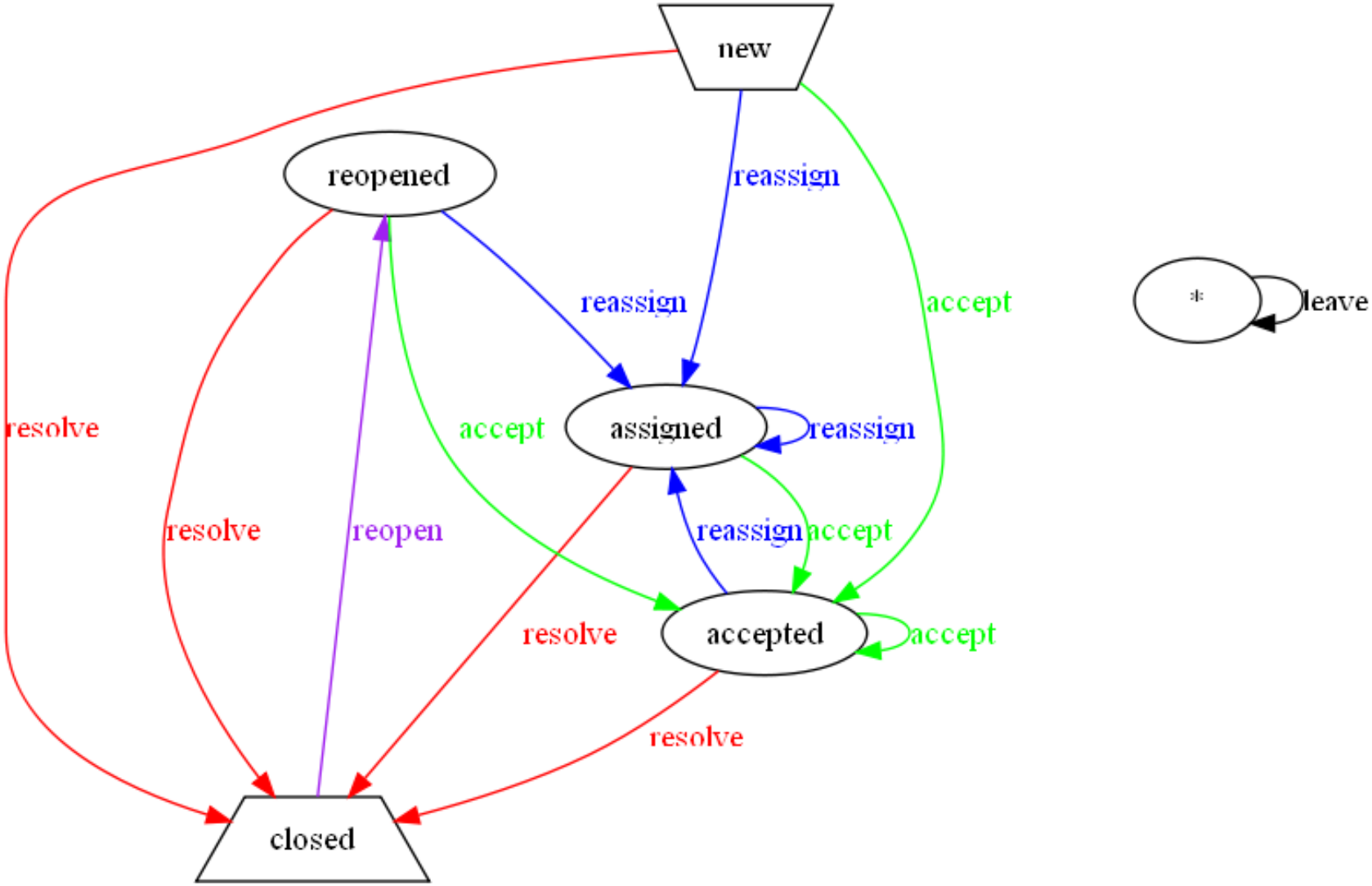
典型的问题处理流程

1. 用户发现问题，创建新 ticket
2. 管理员如果验证该问题合法，则转到③；否则转到 ⑦
3. 确定负责解决该问题的技术人员
4. 技术人员选择接受该 ticket
5. 技术人员解决该问题后，打开该 ticket，描述如何解决的
6. 用户收到关于该问题的解决情况的通知
7. 项目管理员关闭该问题

Trac 的 Ticket

- ID: 在 wiki 中通过 #num 直接引用
- 概述：对问题的一句话概要性描述
- 描述：对该问题的具体描述信息。应可重现
- 类型：是 Bug, Task, Enhancement
- 优先级：Critical, Major, Minor, Trivial
- 版本 -构件：发生该问题的具体版本 -构件
- 里程碑：什么时候该问题应该被解决
- 报告者：提交问题的人
- 属主：谁应该解决该问题
- 抄送：谁应该关注该问题

Trac 的处理流程图


new
reopened
reassign
reassign
accept
resolve
accept
assigned
reassign
resolve
reopen
reassign
accept
resolve
accepted
accept
resolve
closed
*
leave

问题驱动的开发

- 规划项目的组成构件、项目的各个里程碑、版本
- 分解各个里程碑需要完成的任务，并将任务分配到具体的程序员
  - 任务必须是非常明确的、可执行、可测试的
  - 关键任务应有单元测试用例
- 里程碑的所有任务完成后，应该进行一次版本发布，并提供该版本对系统的改动 (ChangeLog)，这通常可自动生成
- 提交到代码库时，在提交说明中应标记此次提交对应的问题
- 每个里程碑应有集成测试用例

常见问题跟踪系统

- Trac
- Redmine
- Bugzilla
- Launchpad
- Bugfree
- JIRA
- ...

内容提要

1 patch

2 版本控制

3 问题跟踪系统

4 Daily Build

Daily Build

每日创建: Daily build, nightly build

1. 将一个软件项目的所有最新代码取出
2. 从头开始编译，链接
3. 生成文档、网页、发布包等
4. 运行测试软件包进行单元测试并对主要功能进行测试
5. 发现错误并报告错误

上述过程完全自动化!

Daily build 的好处

- 及早发现不正确的代码提交
- 及早发现项目中存在的问题
- 对所有的变动进行自动化测试
- 随时可得到可运行的系统
- 减少风险

如何进行 Daily build

需要有若干工具支持

- 源码控制系统: cvs, svn
- 编译工具: make(Makefile), ant(build.xml), maven
- 测试工具: Junit, Nunit
- 编译、运行、测试脚本

利用 Unix 脚本进行 Daily build

可以写一个 Unix 脚本 build.sh, 完成简单的 daily build 功能

- 从 cvs 或 svn 中签出代码
- 调用 make 或 ant 进行自动编译
- 若有单元测试用例, 执行单元测试
- 上述任何步骤有错误都将错误信息通过 email 发送给项目组所有人
- 利用cron设定系统在每日某个时刻自动调用 build.sh 脚本

你有更多期望...

build.sh 脚本能力相对局限. 你可能希望

- 每次有程序员提交了代码, 都自动触发 build 脚本
- 在项目网页上生成自动 build 的统计信息,
- 如果自动 build 出错, 可以
  - 给你的手机、告警设备等发信息
  - 定位错误源, 回退到上一个正确版本
  - 和 Bug 管理系统集成

持续集成的概念

持续集成

持续集成 (Continual Integration) 是一种软件工程实践, 是极限编程 (Extreme Programming) 中很重要的一种活动有若干持续集成工具:

- Cruise Control
- continuum
- bamboo(open source license)
- hudson
- bitten
- buildbot