Principles of Software Construction: Objects, Design, and Concurrency

Version control with git

Christian Kästner  Bogdan Vasilescu

(Adapted from Christopher Parnin/NCSU & Prem Devanbu/UC Davis & Kenneth Anderson/CU Boulder)
Part 1: Design at a Class Level
- Design for Change: Information Hiding, Contracts, Design Patterns, Unit Testing
- Design for Reuse: Inheritance, Delegation, Immutability, LSP, Design Patterns

Part 2: Designing (Sub)systems
- Understanding the Problem
- Responsibility Assignment, Design Patterns, GUI vs Core, Design Case Studies
- Testing Subsystems
- Design for Reuse at Scale: Frameworks and APIs

Part 3: Designing Concurrent Systems
- Concurrency Primitives, Synchronization
- Designing Abstractions for Concurrency
- Distributed Systems in a Nutshell

Intro to Java
Git, CI

UML
Static Analysis
Performance

GUIs

Config management
More Git
Administrivia

• Homework 6 due tonight, 11:59 p.m.

• Final exam Thursday, 11 May, 1pm-4pm in GHC 4401

• Review session in Wednesday, 10 May, 6pm-9pm in GHC 4401
The Modern World

• Which Version?
• How to recreate?
• How to fix?
• Where to apply the fix?
• How/when to Redistribute?
Components of Modern Configuration Management

Version Control: Branches/Forks/Workflow
Task and Build managers
Build machines, virtual environments (dev stacks)
Package managers
Containers, VMs, in the Cloud
Deployment –Infrastructure as Code.
Data migration

Other issues: orchestration, inventory, compliance
Task and Build Managers

Analyze dependencies, and efficiently build (only) what needs to be built or rebuilt.

Tools: make, ivy, ant, maven, gradle, ...
Make dependency graph

- A makefile can be modeled as a dependency graph. The make algorithm performs a traversal over the graph. Each node is checked after all of its children, and the actions are run if any child has a timestamp greater than its parent.
Good things about make

• Available on pretty much every darn platform.
• Very fast.
• Fully featured programming language (but weird)
• First mover advantage
Bad things about make

• Weird syntax (indent is tab, NOT space)
• Has only global variables.
• Where shell can be used, and where make commands? Weird.
• No standards for anything. E.g.,: recursion, dependency analysis, file lists.
• No “reuse” or inheritance of makefiles.
• Not portable across OS, even across Unix flavors.
• Debugging? Yeah, good luck with that.
• Can’t guarantee consistency/reproducibility.
Ant
Ant vs make

• Make file in XML.
• Replace weird indentations with weird angle brackets.
• Replace “variables” with <property />
• Replace “targets” with <target name=“jar” />
• Replace “rules” with <target name =“jar”,
  depends=“init, classes” />
• Replace “recipes in shell” with tasks.
  “<javac />”,
  “<mkdir />”,
  “<jar />”
Ant’s model

• Everything is a Task (sort of)
  – A task has an associated XML element in Ant build files and an associated Java class that implements the task.
  – The XML element can have various attributes and sub-elements, converted into parameters and passed to the Java class.
  – Build file called build.xml by convention
    • First task executed by invoking its associated Java class and passing it its input parameters (if any).

• What’s the difference between tasks as shell commands vs tasks as Java?
Ant Project Format

• build.xml

• Project Name
• Property Values
• Paths
• Tasks
• Targets
Construction of Ant Build Files

• The default name for a Ant build file is build.xml

• The xml root element must be the ‘project’ element
  – The ‘default’ attribute of the project element is required and specifies the default target to use

• Targets contain zero or more AntTasks
  – The ‘name’ attribute is required

• AntTasks are the smallest units of the build process
Ant Build File Example

```
<project default="hello">
  <target name="hello">
    <echo message="Hello, World"/>
  </target>
</project>
```

**Execution of build file:**

```sh
% ant
Buildfile: build.xml
hello: [echo] Hello, World
BUILD SUCCESSFUL
Total time: 2 seconds
```
Ant Build File Example

```xml
<project default="hello">
  <target name="hello">
    <echo message="Hello, World"/>
  </target>
  <target name="bye">
    <echo message="goodbye, World"/>
  </target>
</project>
```

```
franky:xx devanbu$ ant bye
Buildfile: /private/tmp/xx/build.xml
bye:
  [echo] Bye, World
BUILD SUCCESSFUL
Total time: 0 seconds
franky:xx devanbu$ ant hello
Buildfile: /private/tmp/xx/build.xml
hello:
  [echo] Hello, World
BUILD SUCCESSFUL
Total time: 0 seconds
```
Ant Properties

• `<property name="lib.dir" value="lib"/>
• From command line
• In build.xml
• From external XML
• From external property files
• From environment
Ant Path, Ant Target/Task

<path id="classpath">
  <fileset dir="${lib.dir}"
    includes="**/*.jar"/>
</path>

<target name="compile">
  <mkdir dir="${classes.dir}"/>
  <javac srcdir="${src.dir}"
    destdir="${classes.dir}"
    classpathref="classpath"/>
</target>
Ant Target

- Name
- Description
- Dependencies
- Conditionals
- `<antcall>` task
Ant Tasks

• Core Tasks

• Optional Tasks

• Custom Tasks

Dependencies
- Version Compatibility?
- Tracking bug/security fixes?
- Transitive dependencies?
- Consistency?
Imperfect techniques to manage dependencies

• Placing all dependent projects (JAR files) in a directory that's checked into the project's version-control repository.
• Allocating dependent JARs to a common file server
• Copying JAR files manually to a specific location on each developer's workstation.
• Performing an HTTP Get to download files to a developer's workstation, either manually or as part of the automated build.
Ivy
Defining dependencies in ivy.xml

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="./config/ivy/ivy-doc.xsl"?>
<ivy-module version="1.0">
  <info organisation="com" module="integratebutton" />
  <dependencies>
    <dependency name="hsqldb" rev="1.8.0.7" />
    <dependency name="pmd" rev="2.0" />
    <dependency name="cobertura" rev="1.9"/> 
    <dependency name="checkstyle" rev="4.1" />
    <dependency name="junitperf" rev="1.9.1" />
    <dependency name="junit" rev="3.8.1" />
  </dependencies>
</ivy-module>
```

- Note: no indication of file locations or URLs
- Convention: dependency name="cobertura" rev="1.9" translates to cobertura-1.9.jar
Specifying dependencies in Ivy

ivy.xml

```xml
<ivy-module version="2.0">
  <info organisation="org.apache" module="WebProject" />

  <dependencies>
  <dependency org="org.slf4j" name="slf4j-api" rev="1.7.6" conf="compile-default"/>
  <dependency org="jstl" name="jstl" rev="1.2" conf="compile-default"/>
  <dependency org="ch.qos.logback" name="logback-classic" rev="1.1.2" conf="compile-default"/>
  <dependency org="org.springframework" name="spring-core" rev="4.1.3.RELEASE" conf="compile-default"/>
  <dependency org="org.springframework" name="spring-beans" rev="4.1.3.RELEASE" conf="compile-default"/>
  <dependency org="org.springframework" name="spring-context" rev="4.1.3.RELEASE" conf="compile-default"/>
  <dependency org="org.springframework" name="spring-web" rev="4.1.3.RELEASE" conf="compile-default"/>
  <dependency org="org.springframework" name="spring-webmvc" rev="4.1.3.RELEASE" conf="compile-default"/>
  </dependencies>
</ivy-module>
```

- Note: no indication of file locations or URLs
- Convention: dependency name="cobertura" rev="1.9" translates to cobertura-1.9.jar
Ivy settings file

```xml
<ivysettings>
  <settings defaultResolver="chained"/>
  <resolvers>
    <chain name="chained" returnFirst="true">
      <filesystem name="libraries">
        <artifact pattern="${ivy.conf.dir}/repository/[artifact]-[revision].[type]" />
      </filesystem>
      <url name="integratebutton">
        <artifact pattern="http://www.integratebutton.com/repo/[organisation]/[module]/[revision]/[artifact]-[revision].[ext]" />
      </url>
      <ibiblio name="ibiblio" />
      <url name="ibiblio-mirror">
        <artifact pattern="http://mirrors.ibiblio.org/pub/mirrors/maven2/[organisation]/[module]/[branch]/[revision]/[branch]-[revision].[ext]" />
      </url>
    </chain>
  </resolvers>
</ivysettings>
```
Depending on dependencies

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ivy-module version="2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd">
    <info organisation="cobertura" module="cobertura" revision="1.9"/>
    <configurations>
        <conf name="master"/>
    </configurations>

    <publications>
        <artifact name="cobertura" type="jar" conf="master"/>
    </publications>

    <dependencies>
        <dependency org="objectweb" name="asm" rev="2.2.1" conf="master"/>
        <dependency org="jakarta" name="oro" rev="2.0.8" conf="master"/>
        <dependency org="apache" name="log4j" rev="1.2.9" conf="master"/>
    </dependencies>
</ivy-module>
```
<project xmlns:ivy="antlib:org.apache.ivy.ant" name="HelloProject" default="main" basedir="." >
<description>
Create a Spring MVC (WAR) with Ant build script
</description>

<!-- Project Structure -->
<property name="jdk.version" value="1.7" />  
<property name="projectName" value="WebProject" />  
<property name="src.dir" location="src" />  
<property name="resources.dir" location="resources" />  
<property name="web.dir" value="war" />  
<property name="web.classes.dir" location="${web.dir}/WEB-INF/classes" />  
<property name="target.dir" location="target" />  
<property name="target.temp.dir" location="target/temp" />  
<property name="lib.dir" value="lib" />

<!-- ivy start -->
<target name="resolve" description="retrieve dependencies with ivy">
<echo message="Getting dependencies..." />
<ivy:retrieve />

<ivy:cache pathid="compile.path" conf="compile" />
<ivy:cache pathid="runtime.path" conf="runtime" />
<ivy:cache pathid="test.path" conf="test" />
</target>

<!-- install ivy if you don't have ivyide-->  
<target name="ivy" description="Install ivy">
<mkdir dir="${user.home}/.ant/lib" />
<get dest="${user.home}/.ant/lib/ivy.jar" 
src="http://search.maven.org/remotecontent?filepath=org/apache/ivy/ivy/2.4.0-rc1/ivy-2.4.0-rc1.jar" />
</target>

<!-- ivy end -->

<!-- Compile Java source from ${src.dir} and output it to ${web.classes.dir} -->
<target name="compile" depends="init, resolve" description="compile source code">
<mkdir dir="${web.classes.dir}" />
<javac destdir="${web.classes.dir}" source="${jdk.version}" target="${jdk.version}">
debug="true" includeantruntime="false" classpathref="compile.path"/>
</javac>
</target>
Maven
Main Ideas of Maven

• “Convention over Configuration”
• DESCRIBE, don’t IMPLEMENT.
• Reuse build logic & standards whenever possible (mostly done as “Maven plugins”)
• Organize dependencies clearly, logically, aesthetically
Main Benefits of Maven

• Reuse across multiple projects on the same platform.

• Smaller, more standardized, reusable, build procedures.

• Spend less time on Build, more time Coding Apps
A simple Java app

- `mvn archetype:generate -DgroupId=edu.cmu.cs -DartifactId=hello -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false`
  - What’s in the directory structure?

- Mvn compile

- Run: `java -cp target/classes edu.cmu.cs.App`

- Mvn clean

- Mvn test (see test results)
- Mvn package
- Java `–cp target/*jar edu.cmu.cs.App`
Gradle
Task-Based Managers: Gradle

• Combines the best of Ant and Maven

• From Ant keep:
  • Portability: Build commands described platform-independently
  • Flexibility: Describe almost any sequence of processing steps

• ... but drop:
  • XML as build language, inability to express simple control flow

• From Maven keep:
  • Dependency management
  • Standard directory layouts & build conventions for common project types

• ... but drop:
  • XML, inflexibility, inability to express simple control flow
Summary
What every build system must do:

- Manage dependencies within-project.
- Manage dependencies for outside libraries.
- Maintain consistency and versioning.
- Know tasks that “complete” the dependencies.
- Deal with complex directory structures and many types of files.
- Be as simple as possible, but no simpler.

*How do each of the build systems we discussed do at all this?*
Building a project should be repeatable and automated

• All but the smallest projects have a nontrivial build process

• You want to capture and automate the knowledge of how to build your system, ideally in a single command

• Build scripts are code (executable specifications) that need to be managed just like other pieces of code

• Use a build tool to script building, packaging, testing, and deploying your system
  – Most IDEs have an integrated build system
Versioning entire projects
Which files to manage

• All code and noncode files
  – Java code
  – Build scripts
  – Documentation

• Exclude generated files (.class, ...)

• Most version control systems have a mechanism to exclude files (e.g., .gitignore)
COLLABORATION
Collaborating on Files

• How to exchange files
  – Send changes by email
  – Manual synchronization at project meeting
  – All files on shared network directory

• Permission models
  – Each file has an owner; only person allowed to change it
  – Everybody may change all files (collective ownership)
Concurrent Modifications

• Allowing concurrent modifications is challenging
• Conflicts (accidental overwriting) may occur
• Common strategies
  – Locking to change
  – Detecting conflicts (optimistic model)
Change Conflicts

Two users read the same file

Repository

A

Read

Read

A

Harry

Sally

They both begin to edit their copies

Repository

A

A'

Harry

Sally

Harry publishes his version first

Repository

A'

Write

A'

Harry

A''

Sally

Sally accidentally overwrites Harry's version

Repository

A''

Write

A'

Harry

A''

Sally

source „Version Control with Subversion“
Locking Files

Practical problems of locking model?
Locking Problems

• How to lock?
  – Central system vs announcement on mailing list
  – Forgetting to unlock common

• Unnecessary sequentializing
  – Cannot work on different concepts in same file

• False sense of security
  – Changing dependant files can cause conflicts not prevented by locking
Merging (1/2)

Two users copy the same file

They both begin to edit their copies

Sally publishes her version first

Harry gets an "out-of-date" error
Merging (2/2)

Harry compares the latest version to his own

A new merged version is created

The merged version is published

Now both users have each others’ changes
Example

```java
import java.util.LinkedList;

public class Stack<T> implements Cloneable {
    private LinkedList<T> items = new LinkedList<T>();
    public void push(T item) {
        items.addFirst(item);
    }
    public T pop() {
        if(items.size() > 0) return items.removeFirst();
        else return null;
    }
}
```
Example

```java
import java.util.LinkedList;
public class Stack<T> implements Cloneable {
    private LinkedList<T> items = new LinkedList<T>();
    public void push(T item) {
        items.addFirst(item);
    }

    public int size() {
        return items.size();
    }

    public T pop() {
        if(items.size() > 0) return items.removeFirst();
        else return null;
    }
}
```
Example

```java
import java.util.LinkedList;
public class Stack<T> implements Cloneable {
    private LinkedList<T> items = new LinkedList<T>();
    public void push(T item) {
        items.addFirst(item);
    }
    public int size() {
        return items.size();
    }
    public T pop() {
        if(items.size() > 0) return items.removeFirst();
        else return null;
    }
```

System cannot decide order
3-way merge

• File changed in two ways
  – Overlapping changes -> conflicts
  – Merge combines non-conflicting changes from both
• Merging not always automatic
  – diff tool to show changes
  – Manual resolution of conflicts during merge (potentially requires additional communication)
• Automatic merge potentially dangerous
  -> syntactic notion of conflicts
• Merging of binary files difficult
• In practice: most merges are conflict free
BRANCHING
Branching

• Parallel copies of the source tree
• Can be changed independently, versioned separately, and merged later (or left separate)
• Often used for exploratory changes or to isolate development activities
• Many usage patterns, common:
  – Main branch for maintenance OR main development
  – New branches for experimental features; merge when successful
  – New branches for nontrivial maintenance work
  – Branches for maintenance of old versions
Release management with branches

Diagram shows the process of release management with branches, including:

- Release 1
  - Development
  - End of Release 1 development
  - New feature 1 (for Release 2)
  - QA passes - goes alpha
  - Public release
  - Bug fix

- Release 2
  - QA passes - goes alpha
  - Public release
  - Bug fix

- New feature 2 (for Release 2)
- New feature 3 (for Release 3)

Symbols:
- ▲ Project milestone
- ✗ End of branch
- ↑ Create branch/merge changes
Variants and Revisions

- **Revision** replaces prior revision (temporal)
- **Variant** coexists with other variants
- **Version** describes both
- **Release:** Published and named version

<table>
<thead>
<tr>
<th></th>
<th>V1.0</th>
<th>V1.1</th>
<th>V2.0</th>
<th>V3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base system (Windows)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Linux variant</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Server variant</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Extension for customer A</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Extension for customer B</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Semantic Versioning for Releases

• Given a version number MAJOR.MINOR.PATCH, increment the:
  – MAJOR version when you make incompatible API changes,
  – MINOR version when you add functionality in a backwards-compatible manner, and
  – PATCH version when you make backwards-compatible bug fixes.

• Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

http://semver.org/
Variants and Revisions

Customer B - 1.3

Customer A - 1.3
  A-1.3
  A-1.3.0.1
  1.3 fixes

Customer A - 1.4
  1.4 fixes

Core

[Staples&Hill, APSEC’04]
Managing variants

- Branching for variants does not scale well
- Requires special planning or tooling

- Many solutions
  - Configuration files
  - OO polymorphism
  - Preprocessors
  - Build systems
  - DSLs
  - Software product lines
  - ...

```c
/* common parts */
... /* dependent on operating system */
#if (OS == Unix)
...
#elif (OS == VMS)
...
#else
...
#endif...
```
CENTRALIZED VERSION CONTROL (E.G., SVN)
Classes of version control systems

• Systems supporting merging and/or locking
• Local version control
• Central version control
  – Versions stored on central master server
  – Clients synchronize with server (update, commit)
  – CVS (1990), SVN (2004), Perforce, Visual SourceSafe
• Distributed version control
  – Many repositories; synchronization among repositories (push, pull)
  – Git (2005), Mercurial, Bitkeeper, ClearCase
Centralized Version Control

Client (version 5, branch M)

Client (version 5, branch M)

Client (revision 4, branch B)

Server (all versions)

checkout/ update/ commit

access control
Typical work cycle

- Once: Create local workspace
  - `svn checkout`

- Update workspace:
  - `svn update`

- Perform changes in workspace:
  - `svn add`
  - `svn delete`
  - `svn copy`
  - `svn move`

- Show workspace changes:
  - `svn status`
  - `svn diff`

- Revert changes in workspace:
  - `svn revert`

- Update and merge conflicts:
  - `svn update`
  - `svn resolved`

- Push workspace changes to server:
  - `svn commit`
CVS vs. SVN

CVS

- Improvement over RCS in tracking entire directories
- Revision number per file
- Text files (binary files possible)

SVN

- Revision numbers for project
- Atomic commits (commiting multiple files at once)
- Tracking files and directories
- Support renaming
- Tracking of Metadata
DISTRIBUTED VERSION CONTROL (E.G., GIT)
Git

• Distributed version control
• No central server necessary (but possible)
• Local copies of repositories (containing all history)
  – Locally SVN like functionality: checkout, update, commit, branch, diff
• Nonlinear development: each local copy can evolve independently
• Synchronization among repositories (push/fetch/pull)
• Fast local operations (branch, commit, diff, ...)
Overview

- GH
- M2
- M3

- clone, push, pull
- checkout / update
- commit
Distributed Versions

- Versions not globally coordinated/sorted
- Unique IDs through hashes, relationships tracked in successor graph
  - e.g., 52a0ff44aba8599f43a5d821c421af316cb7305
  - Possible to merge select changes (cherry picking)
  - Possible to rewrite the history as long as not shared remotely (git rebase etc)
- Cloning creates copy of repository (including all versions)
  - Tracks latest state when cloned, relevant for updating and merging
  - Normal checkout and commit operations locally
  - Commits don't change original repository
- Fetch and pull get missing versions from remote repository (one or more)
- Push operations sends local changes to remote repository (one or more), given access rights
Example workflow

Kernel developer
- checkout / update
- commit
- edit
- pull & merge

New developer
- clone
- checkout
- commit
- edit
- pull & merge

Linux
- clone / pull
- push
- edit

Linux
- clone
- edit

Linux
- commit
- edit

Linux
- commit
Pull Request:

- Github feature to ask developer to pull a specific change (alternative to sending email);
- Integration with Travis CI.
Forks

Kernel developer

Fork: Github feature to clone repository on Github (own copy with full rights)
Forks

Kernel developer
checkout / update

Caution:
Please to not fork 214 repositories.

214 Collaboration Policy: "Here are some examples of behavior that are inappropriate: Making your work publicly available in a way that other students (current or future) can access your solutions, even if others’ access is accidental or incidental to your goals."

Fork: Github feature to clone repository on Github (own copy with full rights)
Repositories in mustache.js
Git History

2009-12-08 575c2d8 bchesneau@gmail.com (bchesneau@gmail.com) Merge remote branch 'thomo/master' into mthomo

2009-12-07 8315955 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Merge branch 'master' of git://github.com/couchapp/couchapp

2009-12-01 cb22bce8 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Bug: Trailing carriage returns are not stripped.

2009-11-28 c649863 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Merge branch 'master' of git://github.com/couchapp/couchapp

2009-11-27 1418f1a Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Fix wrong couchapp path

2009-11-27 1944cc8 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Merge branch 'master' of git://github.com/couchapp/couchapp

2009-11-27 a879065 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Fix UTF-8 issue on WinXP

2009-11-26 9759820 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Merge branch 'master' of git://github.com/couchapp/couchapp

2009-11-19 0bed4d9 Thomas.Mohaupt@gmail.com (Thomas.Mohaupt@gmail.com) Fix WinXP problem: Attachment name is not UTF-8 encoded

2009-11-17 6a52137 mot.sqa.int (mot.sqa.int) Fix USERPROFILE handling

2009-12-08 3482860 bchesneau@gmail.com (bchesneau@gmail.com) fix hooks and compress hook

2009-12-08 68ec8ec bchesneau@gmail.com (bchesneau@gmail.com) fix compress options search path

2009-12-07 3c89e38 jason.davies@apache.org (jason.davies@apache.org) Generate attachment signatures before processing macros.

2009-12-07 54c3e83 jason.davies@apache.org (jason.davies@apache.org) Merge branch 'master' of git://github.com/couchapp/couchapp

2009-12-04 9b976d9 bchesneau@gmail.com (bchesneau@gmail.com) fix _iter_

2009-12-04 4b8c9ef7 bchesneau@gmail.com (bchesneau@gmail.com) fix typo. thanks to markh.

2009-12-04 b37230e bchesneau@gmail.com (bchesneau@gmail.com) patch from @rmg. thanks!

2009-11-29 232328a bchesneau@gmail.com (bchesneau@gmail.com) bump version number to 0.5.1.

2009-11-29 d8c3709 bchesneau@gmail.com (bchesneau@gmail.com) add update template to generators

2009-11-28 0954063 bchesneau@gmail.com (bchesneau@gmail.com) couchapp standalone for macosx via py2app

2009-11-28 4d0f8a6 bchesneau@gmail.com (bchesneau@gmail.com) make sure octent type is defined

2009-11-28 1805363 benoitc@.none (benoitc@.none) new fixes while testing

2009-11-28 056e770 bchesneau@gmail.com (bchesneau@gmail.com) rethink hook system. final version. Now like extensions each hooktype is a list of

2009-11-28 c7771bf bchesneau@gmail.com (bchesneau@gmail.com) rethink extensions. So now extensions are a list of key-value pair. Where key is

2009-11-27 53ae4a0 bchesneau@gmail.com (bchesneau@gmail.com) fix compress extension & load extensions like we load hooks

2009-11-27 348e62d bchesneau@gmail.com (bchesneau@gmail.com) fix template paths with windows

2009-11-27 bfa200c bchesneau@gmail.com (bchesneau@gmail.com) backport import_module from python 2.7

2009-11-27 338e9a4 benoitc@.none (benoitc@.none) more windows fix

2009-11-27 8a39b41 bchesneau@gmail.com (bchesneau@gmail.com) more fixes
Git and Central Repositories

© Scott Chacon “Pro Git”
Social Coding

Eileen M. Uchitelle
eileencodes

Platform Systems Team @GitHub & @Rails Core Team

Overview
Repositories 81  Stars 199
Followers 519  Following 20

Pinned repositories

rails/rails
Ruby on Rails
- Ruby  ★ 35.5k  ▼ 14.5k

rack/rack
a modular Ruby webserver interface
- Ruby  ★ 3.1k  ▼ 1.2k

eileencodes.github.io
Code for eileencodes.com
- CSS  ★ 4  ▼ 2

security_examples
Rails application to demo CSRF, XSS, and XXE vulnerabilities and how to avoid them
- Ruby  ★ 34  ▼ 3

integration_performance_test
Application to test performance of integration versus controller tests
- Ruby  ★ 28  ▼ 7

basecamp/bc3-api
API documentation for Basecamp 3
-  ★ 152  ▼ 24

1,176 contributions in the last year

Learn how we count contributions.
Git Internals

Untracked

Add the file

Remove the file

Unmodified

Edit the file

Stage the file

Modified

Commit

Staged

© Scott Chacon “Pro Git”
Git Internals

© Scott Chacon “Pro Git”
Git Internals

© Scott Chacon “Pro Git”
Git Internals

refs/heads/master
  1a410e third commit
  b
  test.txt
  new.txt
  3c4e9c tree

refs/heads/test
  cac0ca second commit
  test.txt
  new.txt
  0155eb tree

  fa49b0 "new file"
  test.txt
  "version 2"

  b
  test.txt

  d8329f tree
  test.txt
  83baae "version 1"
Summary

• Version control has many advantages
  – History, traceability, versioning
  – Collaborative and parallel development
• Locking vs. merging and merge conflicts
• Collaboration with branches
• From local to central to distributed version control
Lessons (reprise)

- Keep it simple
- Use all the tools you know:
  - A good IDE
  - Static analysis tools like FindBugs
  - Verification tools for critical code
  - Unit tests
  - Assert statements for known invariants
  - Code review for all code intended for other developers or users
  - Continuous integration testing for any project with multiple developers