

# Understanding Git with Alloy

## Milestone 3

Cláudio Lourenço   Renato Neves

University of Minho  
Formal Methods in Software Engineering

July 11, 2012



# Table of contents

Git as VCS

Project Motivation and Objectives

Git Internals

Specification of Operations

Documentation

Conclusion



# Git as VCS

## Git is one of many Version Control Systems

- Fast
- Efficient
- Oriented to snapshots - Not differences
- Widely used



# Project Motivation

## Gap in the description of Git

- Lack of precise descriptions
- Contradictions between manuals

## An opportunity appears

- Git users could benefit from a manual that is precise and rigorous



# The dark world of Git

## The common (and above average) knowledge of Git

- "if there are any uncommitted changes when you run git checkout, Git will behave very strangely." <sup>1</sup>
- "When you create a branch, it will contain everything committed on the branch you created it from at that given point. So if you commit more things on the master branch like you have done (after creating b), then switch to branch b, they won't appear. This is the correct behavior. Does that answer your question?" <sup>2</sup>

---

<sup>1</sup>"Understanding Git" Manual

<sup>2</sup>An user of Git development mailing list



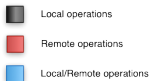
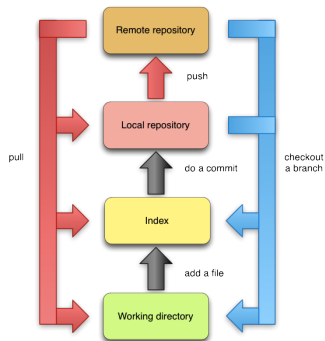
# Objectives of this project

## Shine some light into Git internals

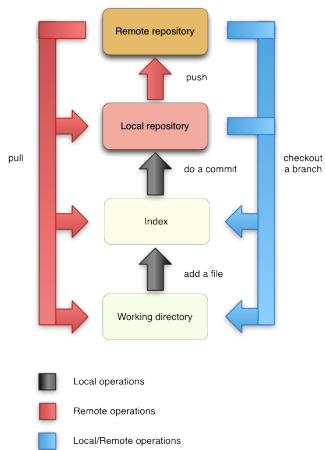
- Build a precise model of how Git works using Alloy
- Analyze the model and verify which properties it does (not) guarantee
- Help others understanding Git, building a manual



# The Git Structure

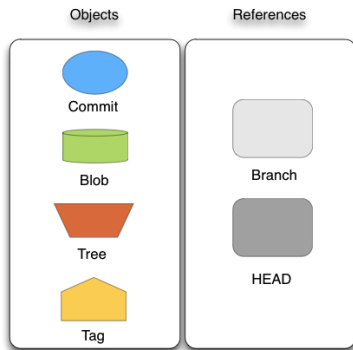


# Repository





# Repository



# Blob and Tree

## Blob

- Represents the content of a file
- The identifier is calculated from its content
- Files with same content will be represented by the same blob

```
sig Blob extends Object {}
```

## Tree

- Contains Blobs or other Trees
- Used to represent the file system structure
- The identifier is calculated in a similar way to the Blobs
- Sharing is done as in Blobs - Same content, same Tree

```
sig Tree extends Object {  
  contains: Name -> lone(Tree+Blob)  
}
```



# Commit

- A snapshot of the project on a certain moment in time
- It has a set of parents, that are considered the previous commits
- Points to a Tree that represents the root folder in the Working Directory
- An auxiliary abstract relation was specified to simplify the commit operation

```
sig Commit extends Object {  
  points : Tree,  
  parent : set Commit,  
  abs: Path  $\rightarrow$  Object,  
  merge : set State  
}
```

```
sig RootCommit extends Commit {}
```



# Branch and HEAD

## Branch

- It is just a pointer to a commit - Unlike other VCS, where a branch is a full copy of the repository

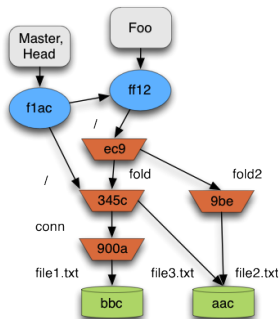
## HEAD

- Special reference that identifies the current Branch

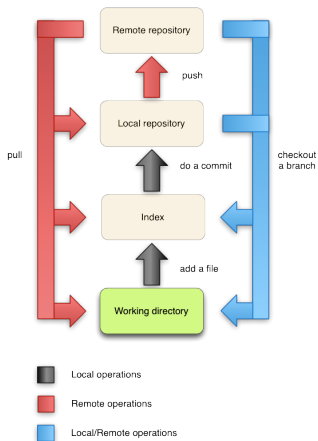
```
sig Branch{  
  marks: Commit lone → State ,  
  branches: set State ,  
  head: set State  
}  
  
lone sig Master extends Branch{}
```



# Repository



# Working Directory



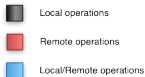
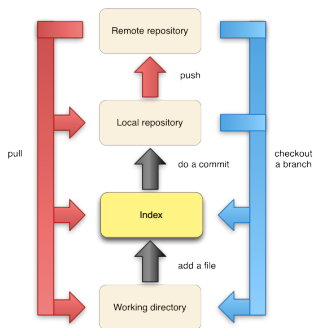
# Working Directory

- Subset of a file system that is tracked by Git
- In our model we simplify this component using the notion of path
- Our convention dictates that if a file exists in an Alloy instance, then it automatically exists in the Working Directory

```
sig Path {  
  pathparent: lone Path,  
  name: Name,  
  unmerge: set State  
}  
  
one sig Root extends Path{}
```



# Index





# Index

- Contains all files that are going to be committed on the next commit
- The index is not necessarily equal to the Working Directory
- If an user wants to commit a new file or a modified file, first he must add it to the index

```
sig File{  
  path: Path,  
  blob: Blob,  
  index: set State  
}
```



# Modeled Operations

- Add and Remove
- Commit
- Branch and Branch Remove
- **Checkout**
- **Merge (2-way and fast-forward)**



# Add and remove

## Add

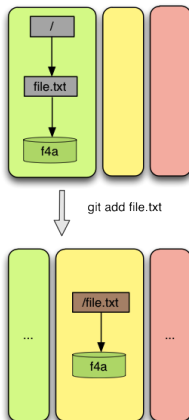
- Adds a file to the index
- Updates the content of a file in the index
- Those changes will be stored in the next commit

## Remove

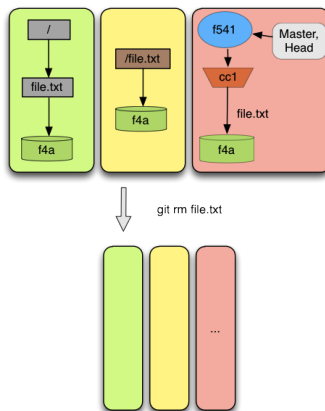
- Removes a file from index, and from the Working Directory
- The removed file will not be present in the next commit



# Add



# Remove



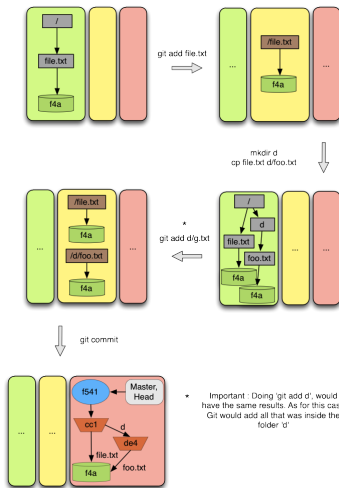
# Commit

## Commit

- A snapshot of a project on a certain moment in time
- In the first version of the model it was very difficult to model such operation
- An abstract relation was then modeled to map objects on a certain commit with a path

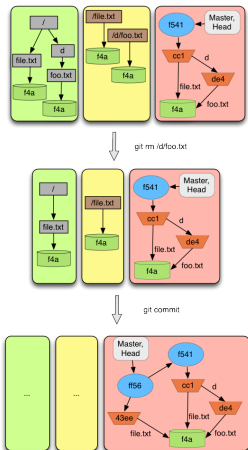


# Commit



\* Important: Doing 'git add d', would have the same results. As for this case Git would add all that was inside the folder 'd'.

# Commit





# Checkout

## The most difficult operation to specify

- No manual was found with a full description
- No references to the existing pre-conditions - "if there are any uncommitted changes when you run git checkout, Git will behave very strangely. The strangeness is predictable and sometimes useful, but it is best to avoid it..."<sup>3</sup>
- Bug was found and reported to the Git Mailing List<sup>4</sup>

---

<sup>3</sup><http://www.sbf5.com/~cduran/technical/git/git-2.shtml>

<sup>4</sup>[git@vger.kernel.org](mailto:git@vger.kernel.org)



# Checkout pre-conditions

## Pre condition expected

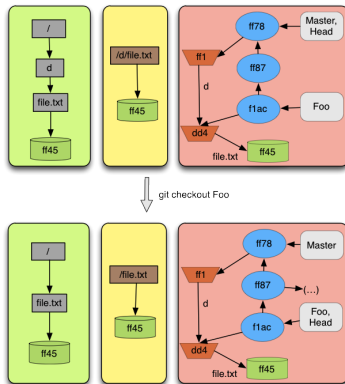
- No uncommitted files

## Pre conditions found

- Everything that is in the index has to be in the current commit with the same content, except if
  - The content of a file is the same in the current and destination commit (warning is thrown)
  - Exists a file in the index, and that file does not exist neither in the current nor in the destination commit (warning is thrown)
  - Content of the file in the index is the same as in the destination commit (no warning is thrown)



# Checkout



# Merge - The Last Operation Modeled

## Merge

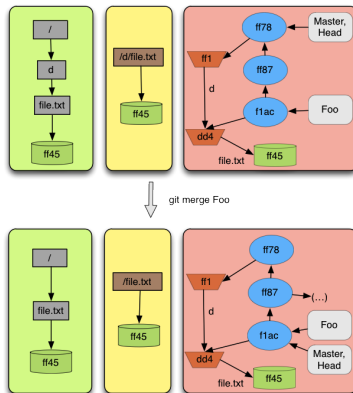
- The current commit and the commit pointed by the specified branch will be merged into a possible new commit

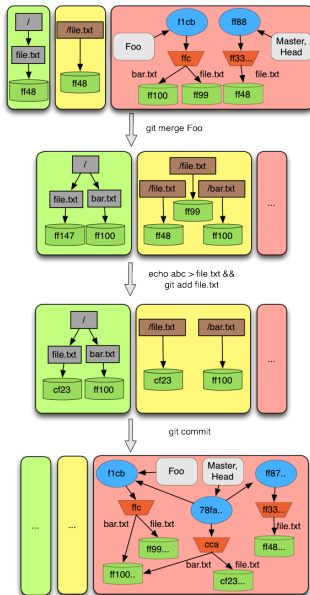
## Merge Tactics

- The commit being merged is more recent than the current commit - It is not created a new commit because the commit being merged takes precedence - **Fast-Forward Merge**
- The typical **2-way merge** will create a new commit resulting from both commits
- The **3-way merge** will also create a new commit, however it uses the most recent common ancestor to solve automatically some conflicts that not solved with 2-way merge

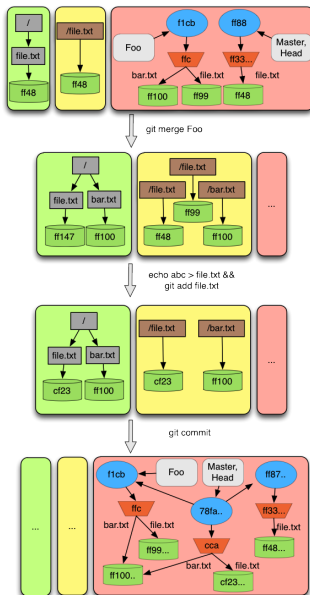


# A fast-forward Merge





# A 3-way Merge



# Merge specification

- Fast-Forward Merge and 2-way Merge are modeled
- 3-way Merge partially modeled
- Two relations had to be created for the case a manual merge takes place:
  - Unmerged Files
  - A relation to identify the branch being merged





# Merge specification

## Most important pre-conditions specified

- The current commit cannot be more recent than the commit being merged
- There cannot be an unfinished merge
- When the fast-forward tactic is applied, the index cannot have uncommitted files that would conflict with files from the resulting merge
- When a 2-way or 3-way merge is applied, it is not possible to have uncommitted files in the index



# Website

- A manual was created using the knowledge obtained
- Website was created based on the manual
- Project documentation is also available
- [http://nevrenato.github.com/CSAIL\\_Git](http://nevrenato.github.com/CSAIL_Git)



## Future Work

- Model some more operations: 3-way merge, rebase, fetch, etc.
- Check for some more properties that the model does (not) guarantee
- Build interactive diagrams of concrete examples of operations



# Conclusions

- Not enough time (not even close) to specify the whole Git
- It would be nice to have completeness in the model verification
- Model is stable and documented enough to be extended with other operations by outsiders

