

Understanding Git with Alloy

Milestone 3

Cláudio Lourenço Renato Neves

University of Minho
Formal Methods in Software Engineering

July 10, 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



Motivation for this project

Gap in the understanding of Git

- Lack of precise descriptions
- Contradictions in some manuals

An opportunity appears

- Developers 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." ¹
- "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?" ²

¹"Understanding Git" Manual

²An user of Git development mailing list



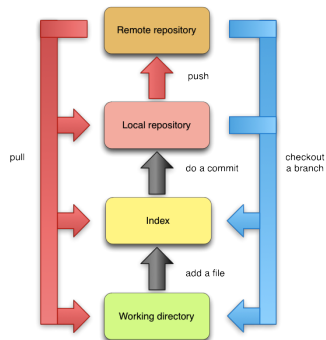
Objectives of this project




Shine some light in Git internals

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



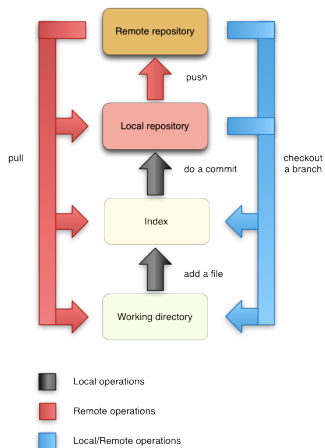
The Git Structure



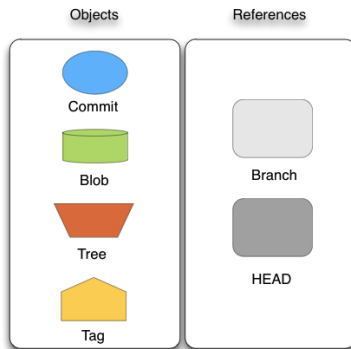
-  Local operations
-  Remote operations
-  Local/Remote operations



Repository



Repository



Blob and Tree

Blob

- Represents the content of a file
- The identifier is calculated from its content, thus files with same content will share 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. Thus, for Trees sharing also exists

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



Commit

- A snapshot of the project on a certain moment in time
- Has a set of parents, that are considered the previous commits
- Points to a Tree that is equivalent to the root folder in the working directory
- An auxiliar relation named "Abstract" was specified to be easy to model 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

- Pointer to a commit, unlike other VCS where a branch is a full copy of the repository

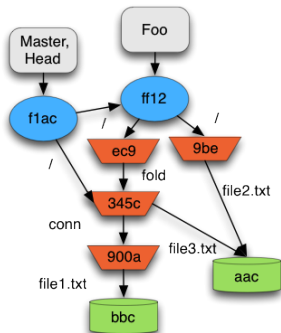
HEAD

- Special reference that identifies the current Branch

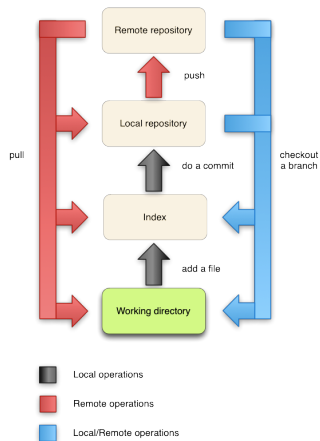
```
sig Branch{  
  marks: Commit lone → State ,  
  branches: set State ,  
  head: set State  
}  
  
lone sig Master extends Branch{}
```



Repository



Working Directory



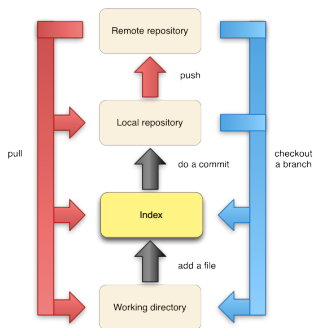
Working Directory




- Subset of a file system, that is tracked by Git
- Was not modeled in this project
- However files were modeled.
- Our convention dictates, if a file exists on an Alloy instance, then automatically exists in the Working Directory

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



Index



-  Local operations
-  Remote operations
-  Local/Remote operations



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 newly 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

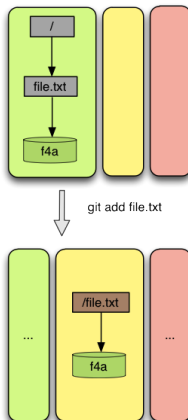
- Adds a file to the index
- Updates the content of a file also in the index
- Those changes will be stored in the next commit

Remove

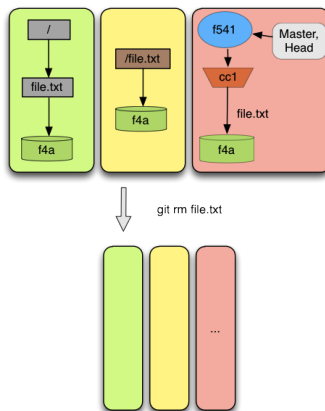
- Removes a file of the index, and from the Working Directory (if it's still there)
- In the next commit the file deleted will not appear



Add



Remove



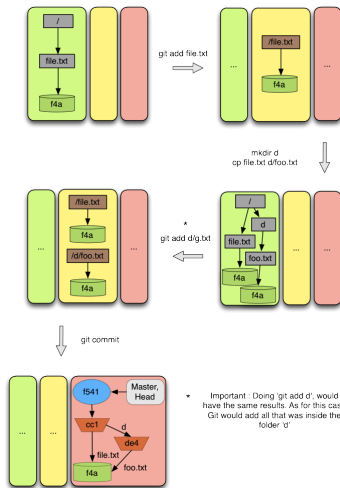
Commit

Commit

- A snapshot of a project on a certain moment in time
- In the first version of the model it was getting almost impossible to specify the operation
- The latest version included a relation that associated the objects in a commit with the correspondent paths

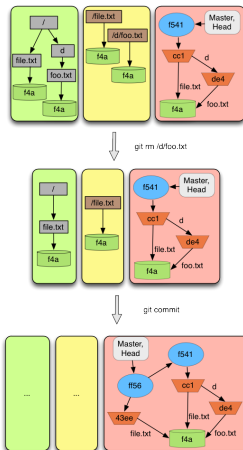


Commit



* 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 that fully described checkout
- Expected pre-conditions were not found. Founding instead weaker pre conditions
- Strange behaviour caught. Discussed about it with the Git mailing list. Concluded that was a bug



Checkout pre-conditions

Pre condition expected

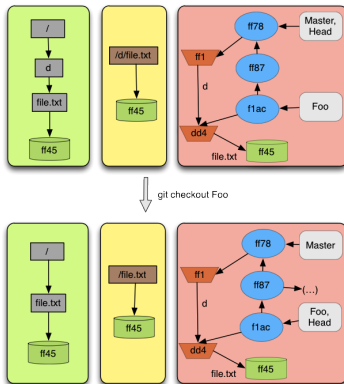
- No uncommitted files

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

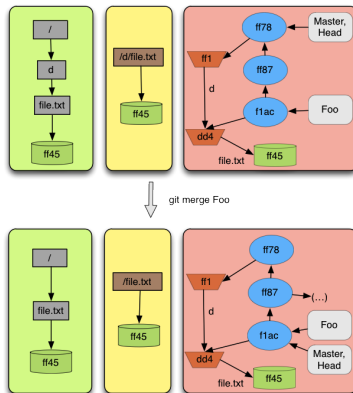
- The current commit and the commit pointed by the specified branch will, be merged into a possibly a new commit

Merge types

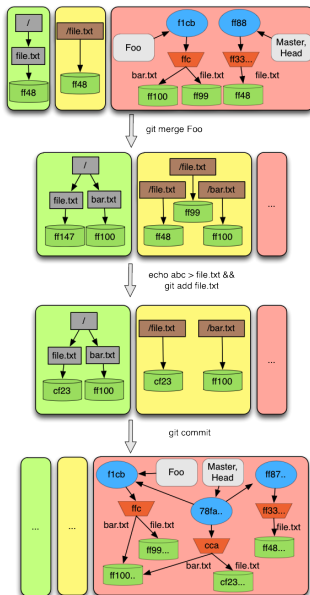
- The commit pointed by the branch, is more recent then the current, thus the newest commit will take always precedence
- The typical 2-way merge, that will create a new commit resulting from both commits specified
- The 3-way merge, that will also create a new merge commit. However this type has much less potential for merge conflicts then the last type



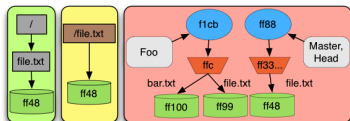
A fast-forward Merge



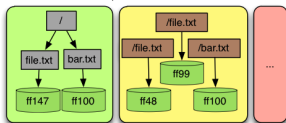
A 2-way Merge



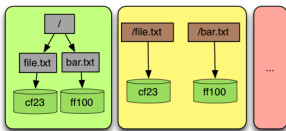
A 3-way Merge



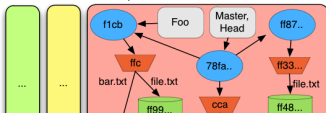
git merge Foo



echo abc > file.txt &&
git add file.txt



git commit



Merge specification

- Modeled the 2-way and fast-forward merge
- Not enough time for 3-way, however there is a semi-tested version
- Created two new relations
 - Which files are unmerged
 - Which commits are going to be parents in the next commit



Merge specification

Most important pre-conditions specified

- The current commit cannot be more recent than the commit pointed by the branch specified
- There cannot be an unfinished merge
- When it's a fast-forward case, the index cannot have uncommitted files that would conflict with files from the resulting merge
- When it's a 2-way or 3-way merge, it is not possible to exist uncommitted files in the index



Website

- A manual was created using the knowledge obtained
- Website was created based on the manual
- Also, all project documentation was put available
- http://nevrenato.github.com/CSAIL_Git



Future Work

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



Conclusions

- Not near enough time to specify all Git operations
- It would be nice to have completeness in the model verification
- Model stable and documented enough to be extended with other operations by outsiders
- Manuals are difficult to do

