COMP0104 Software Development Practice: Distributed Version Control (GIT)

Jens Krinke

Centre for Research on Evolution, Search & Testing
Software Systems Engineering Group
Department of Computer Science
University College London

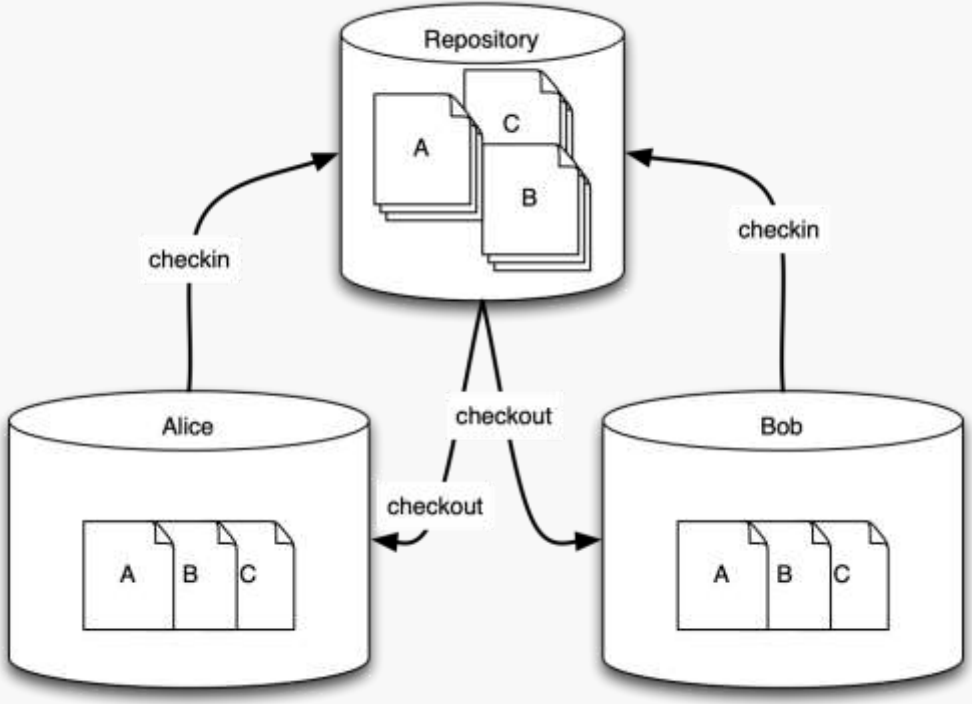
GIT

- Comparison with SUBVERSION
- Distributed Version Control
- Basic Work Cycle
- How GIT works

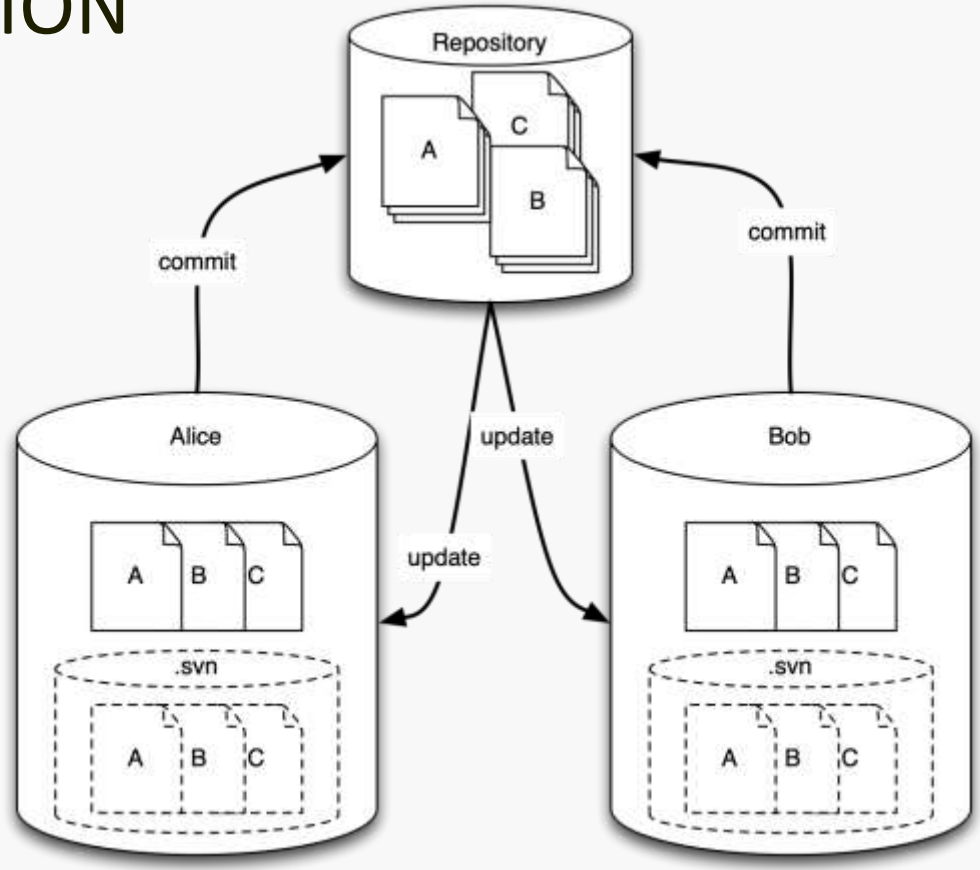
SUBVERSION: Basic Work Cycle

- Update your working copy
  svn update
- Make changes
  svn add/delete/copy/move
- Examine your changes
  svn status/diff
- Resolve conflicts
  svn update/resolve
- Commit your changes
  svn commit

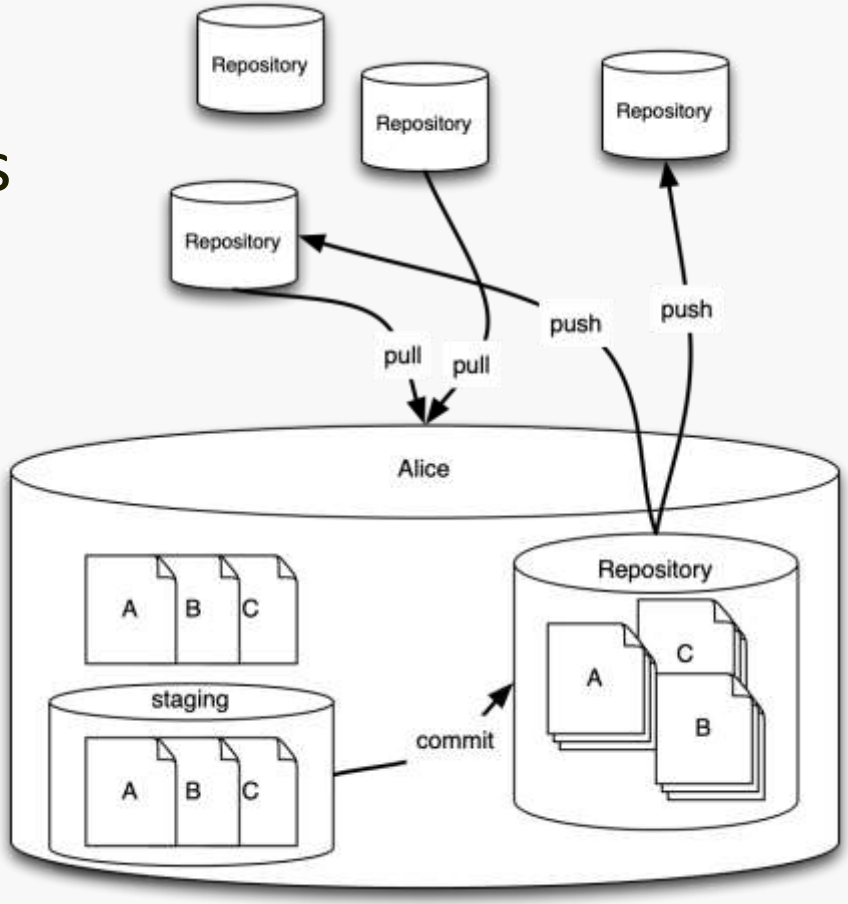
CVS



SUBVERSION



GIT: Distributed Repositories



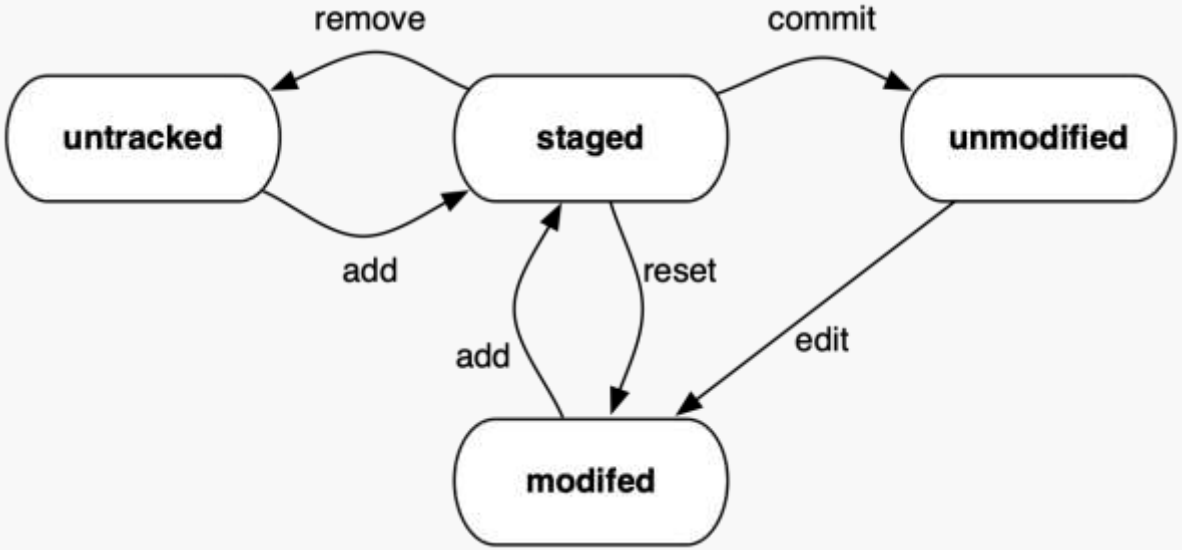
Distributed Version Control

- Almost everything is local
- Almost every operation is fast
- Almost every cloned repository is a backup
- Almost every operation works offline
- Almost all repositories have the full history

GIT's Special: The Staging Area

- Unlike other version control systems, GIT has a concept of the staging area (index)
- These are changes which will be committed.
- The staging area can be used to prepare for the commit.

File Status Lifecycle



Setup

Introduce yourself to GIT:

```
$ git config --global user.name “John Doe”
$ git config --global user.email “j.doe@ucl.ac.uk"
$ _
```

The first repository

- % **git init comp0104**
  Initialized empty Git repository in /home/doe/comp0104/.git/
- % **cd comp0104**
- % **git status**
  # On branch master
  #
  # Initial commit
  #
  nothing to commit (create/copy files and use "git add" to track)

Create a File

File Status: untracked

- % **vi README**
- % **git status**

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will
be committed)
#
#   README
nothing added to commit but untracked files present
(use "git add" to track)
```

Add a File to the Stage
File Status: staged (added)

- % **git add README**
- % **git status**

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   README
#
```

Commit the Changes
File Status: unmodified

- `% git commit -m 'Initial commit.'`

```
[master (root-commit) 8a5e2f0] Initial commit.
 1 file changed, 1 insertion(+)
 create mode 100644 README
```

- `% git status`

```
# On branch master
nothing to commit (working directory clean)
```

File Status: modified

- % **vi be**
- % **vi README**
- % **git add be**

File Status: modified

- ```
  % git status
  # On branch master
  # Changes to be committed:
  #   (use "git reset HEAD <file>..." to unstage)
  #
  #  new file:   be
  #
  # Changes not staged for commit:
  #   (use "git add <file>..." to update what will be
  committed)
  #   (use "git checkout -- <file>..." to discard changes
  in working directory)
  #
  #  modified:   README
  #
  ```
  ```

git diff
Differences between working area and stage

- % **git diff**

```
diff --git a/README b/README
index 9dd26d5..e0f254c 100644
--- a/README
+++ b/README
@@ -1 +1,2 @@
 This is an example project for GIT.
+It contains a file with british english text.
```

- % _

The Stage is not final (status before a change)

- % **git add README**
- % **git status**

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   README
#   new file:   be
#
```

The Stage is not final (change)

- % **vi README**
- % **git diff**

```
diff --git a/README b/README
index e0f254c..5e53579 100644
--- a/README
+++ b/README
@@ -1,2 +1,2 @@
 This is an example project for GIT.
-It contains a file with british english text.
+It contains a file with a british english text.
```

The Stage is not final (status after change)

- ```
  % git status
  # On branch master
  # Changes to be committed:
  #   (use "git reset HEAD <file>..." to unstage)
  #
  #   modified:   README
  #   new file:   be
  #
  # Changes not staged for commit:
  #   (use "git add <file>..." to update what will be
  committed)
  #   (use "git checkout -- <file>..." to discard changes
  in working directory)
  #
  #   modified:   README
  #
  ```
  ```

Accessing stage and repository
stage : repository

- % **git diff --staged README**

```
diff --git a/README b/README
index 9dd26d5..e0f254c 100644
--- a/README
+++ b/README
@@ -1 +1,2 @@
 This is an example project for GIT.
+It contains a file with british english text.
```

- % _

Accessing stage and repository workspace : stage

- % **git diff README**

```
diff --git a/README b/README
index e0f254c..5e53579 100644
--- a/README
+++ b/README
@@ -1,2 +1,2 @@
 This is an example project for GIT.
-It contains a file with british english text.
+It contains a file with a british english text.
```

- % _

Accessing stage and repository workspace : repository

- % **git diff -r master README**

```
diff --git a/README b/README
index 9dd26d5..5e53579 100644
--- a/README
+++ b/README
@@ -1 +1,2 @@
 This is an example project for GIT.
+It contains a file with a british english text.
```

- %

git commit

- % **git commit**

```
Introduced british english.
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README
#       new file:   be
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#       modified:   README
#
```

git log

- % **git log**

```
commit 5f43cee4b5ad7684477828c1a3b8b84c3d345539
Author: John Doe <john@doe.org>
Date:   Sat Jan 12 17:42:44 2013 +0000

    Introduced british english.

commit 8a5e2f02ff6ff4fec8cadd0c4028ffe0f822a6e5
Author: John Doe <john@doe.org>
Date:   Sat Jan 12 17:00:02 2013 +0000

    Initial commit.
```

- % _

Working with multiple repositories (local)

- `% cd ..`
- `% git clone comp0104 comp0104us`
  ```
  Cloning into 'comp0104us'...
  done.
  ```
- `% cd comp0104us`
- `% git status`
  ```
  # On branch master
  nothing to commit (working directory clean)
  ```

Skipping staging

- % **vi be**
- % **git commit -a -m 'Created american version.'**

```
[master 0b5a44f] Created american version.
 1 file changed, 4 insertions(+), 2 deletions(-)
```

- % **vi README**
- % **git commit -a -m 'Updated README file.'**

```
[master f422033] Updated README file.
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Renaming files

- % **git mv be ae**
- % **git commit -m 'Changed file name.'**

```
[master 266abb7] Changed file name.
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename be => ae (100%)
```

Working with multiple repositories

- % **cd ../comp0104**
- % **git status -s**

```
 M README
```

- % **git add README**
- % **git commit -m 'Improved spelling.'**

```
[master febe05c] Improved spelling.
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Working with multiple repositories
git pull

- % **cd ../comp0104us**
- % **git pull**

```
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /home/doe/comp0104
   5f43cee..febe05c  master     -> origin/master
Auto-merging README
CONFLICT (content): Merge conflict in README
Automatic merge failed; fix conflicts and then
commit the result.
```

Resolving conflicts

- % **vi README**

```
This is an example project for GIT.
<<<<<<< HEAD
It contains a file with american english text.
=======
It contains a file with a british english text.
>>>>>>>
febe05cff0734d7dda529fbd761b599297bb86c9
```

- % **git add README**
- % **git commit -m 'Integrated changes.'**
  [master aed3adc] Integrated changes.

History

- **% git log --graph --decorate --pretty=oneline --abbrev-commit**

```
*   aed3adc (HEAD, master) Integrated changes.
|\
| * febe05c (origin/master, origin/HEAD) Improved spelling.
* | 266abb7 Changed file name.
* | f422033 Updated README file.
* | 0b5a44f Created american version.
|/
* 5f43cee Introduce
* 8a5e2f0 Initial c
```

- %

Cloning a remote repository

- % **git clone https://github.com/jkrinke/comp0104gitexmp.git**

```
Cloning into 'comp0104gitexmp'...
remote: Counting objects: 10, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 10 (delta 0), reused 10 (delta 0)
Unpacking objects: 100% (10/10), done.
```

- % **cd comp0104gitexmp**
- %

Updating a remote repository

- % **vi README**
- % **git commit -a -m 'Added a reference to UCL.'**

```
[master 981374f] Added a reference to UCL.
 1 file changed, 1 insertion(+)
```

- % **git push**

```
Username for 'https://github.com': jkrinke
Password for 'https://jkrinke@github.com':
To https://github.com/jkrinke/comp0104gitexmp.git
   febe05c..981374f  master -> master
```

Updating local repository

- % **git pull**

```
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1)
Unpacking objects: 100% (3/3), done.
From https://github.com/jkrinke/comp0104gitexmp
   febe05c..981374f  master     -> origin/master
Updating febe05c..981374f
Fast-forward
 README |    2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Pull

- Adding the changes from other repositories to the local repository.
- Merges are usually automatic.
- Conflicts will occur, but relatively easy to deal with.

Push

- Propagating (local) changes to other repositories.
- Not all changes need to be pushed.
- Push can be replaced by pull.
- Propagating outside push/pull:
  - GIT can create patches
  - GIT can integrate patches

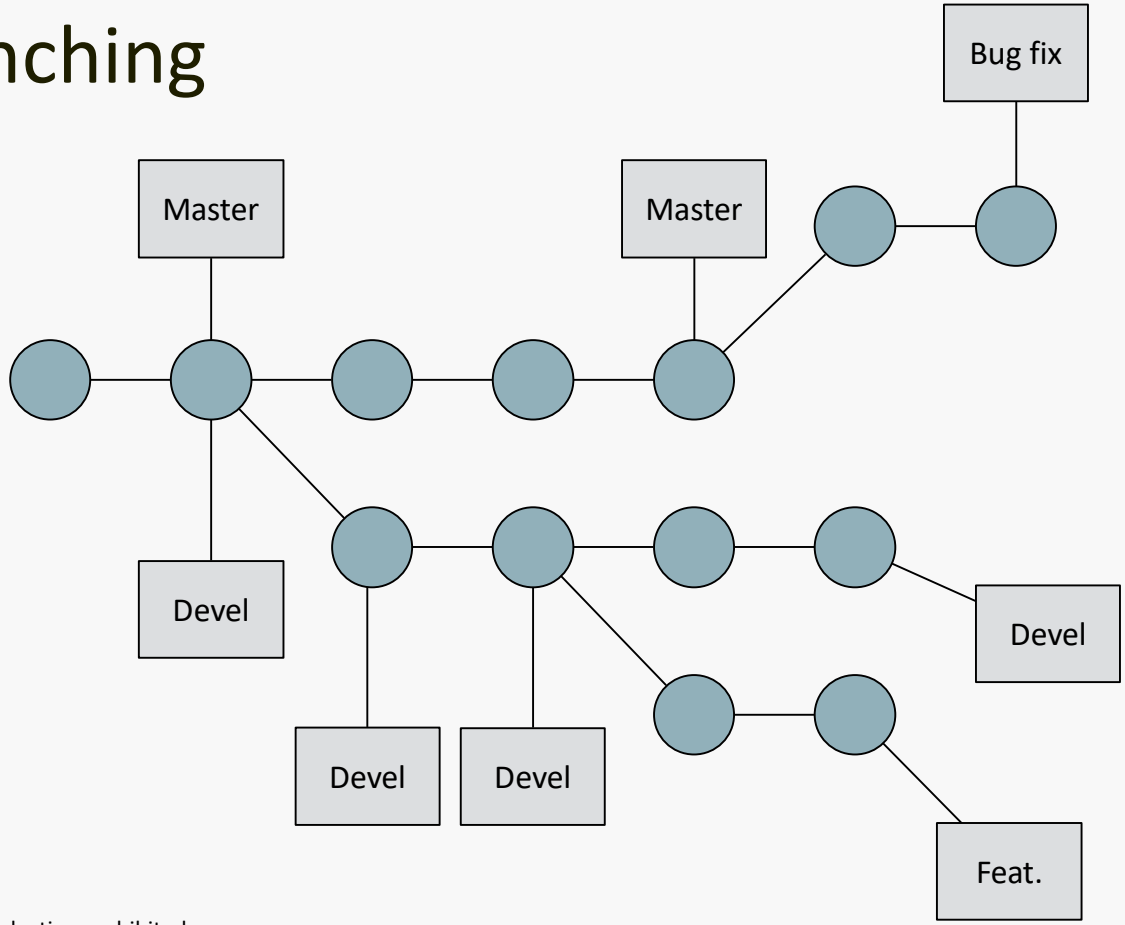
Branching and Merging

- Branching and merging is simple in git.
- Branches of often temporary.
- Branching is encouraged.
- Merging is mainly automatic.
- However, branching can cause problems due to conflicts at merge.
- Resolving conflicts without tool support is painful.
- One should adapt a defined workflow.

Creating Branches
git branch ***name***

- Branches are created from the current branch.
- To commit to a branch, it has to be checked out (git checkout *name*)
- If a branch is no longer needed, it can be deleted.
- Checkout a branch to switch to it in the same workspace.

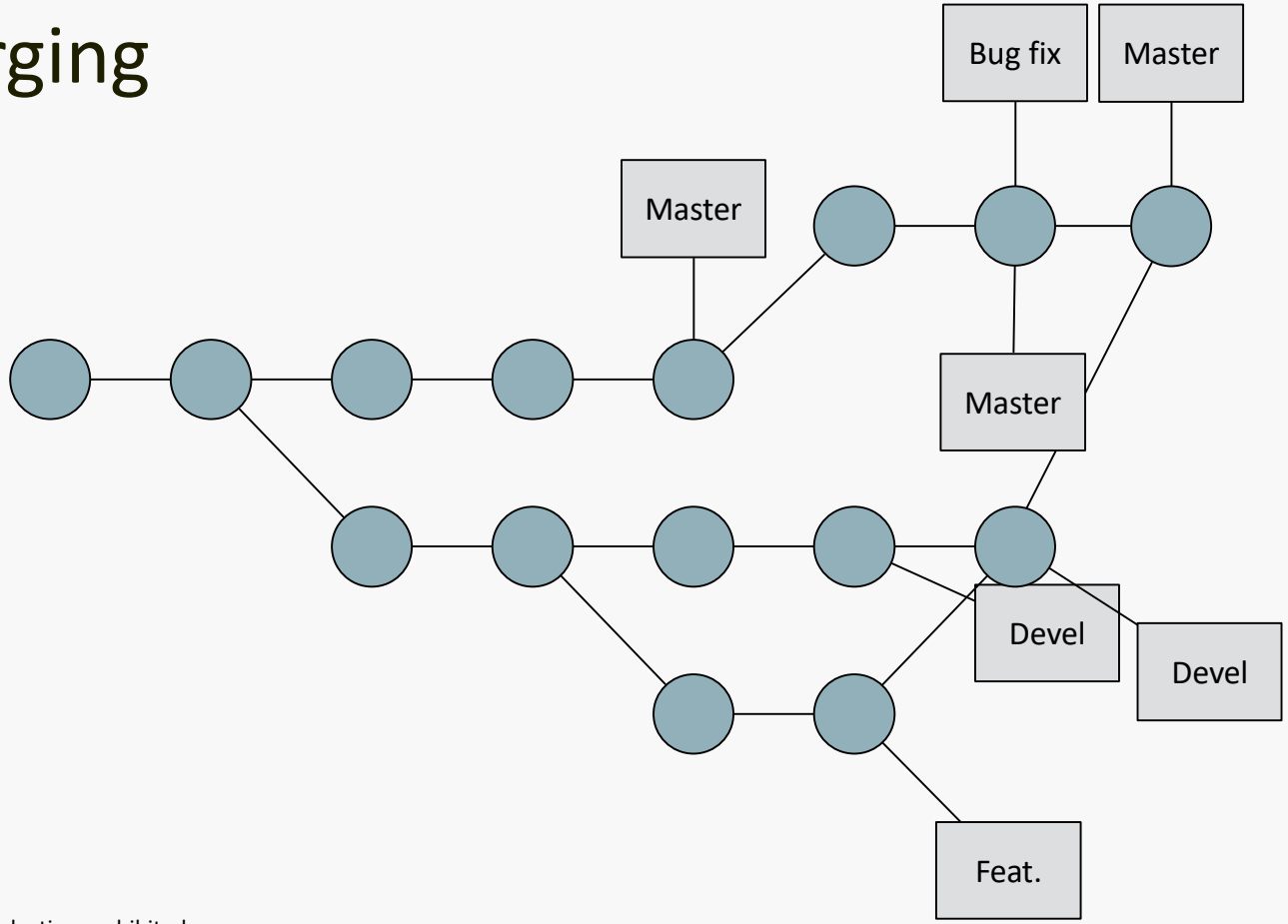
Branching



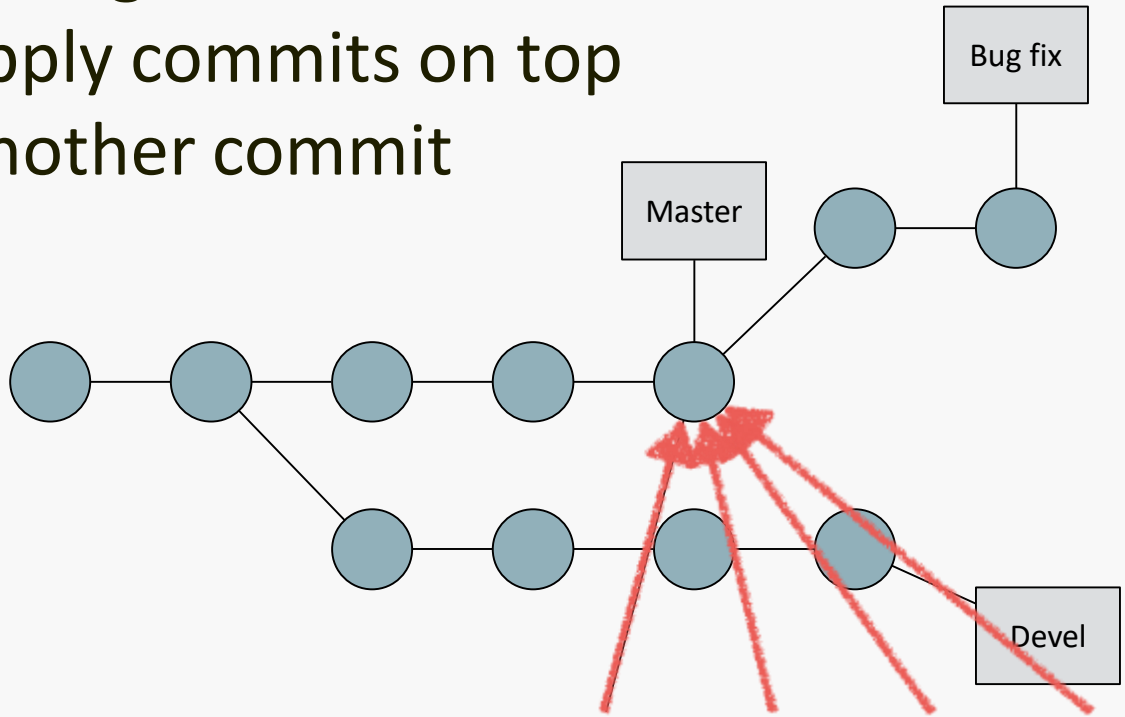
Merging Branches
git merge ***name***

- A branch can be merged into the current one via fast-forward or 3-way merge.
- The current branch will contain the merge result.
- 3-way merge occurs between the current state of the two branches and the common ancestor.
- A fast-forward is only possible if the current branch has not been changed.

Merging

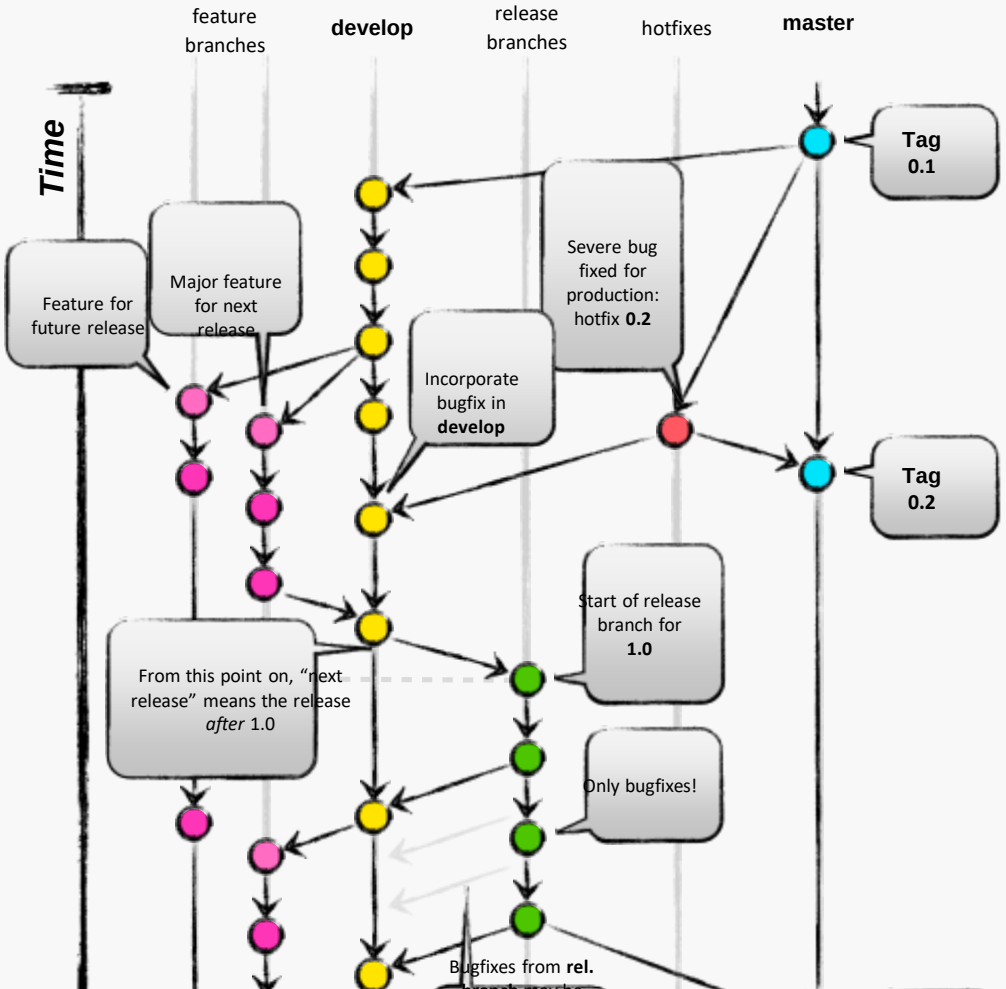


Rebasing:
Reapply commits on top of another commit

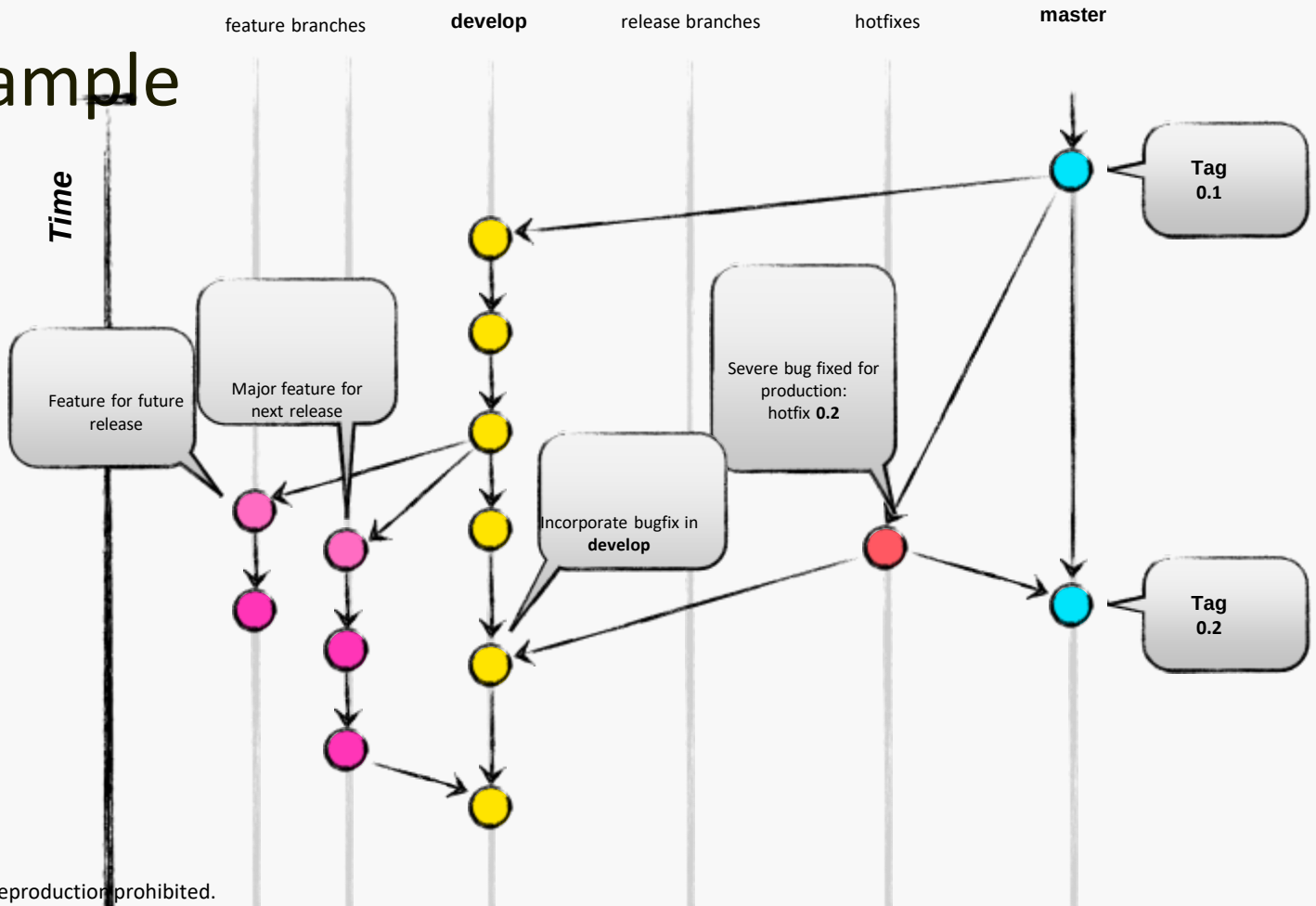


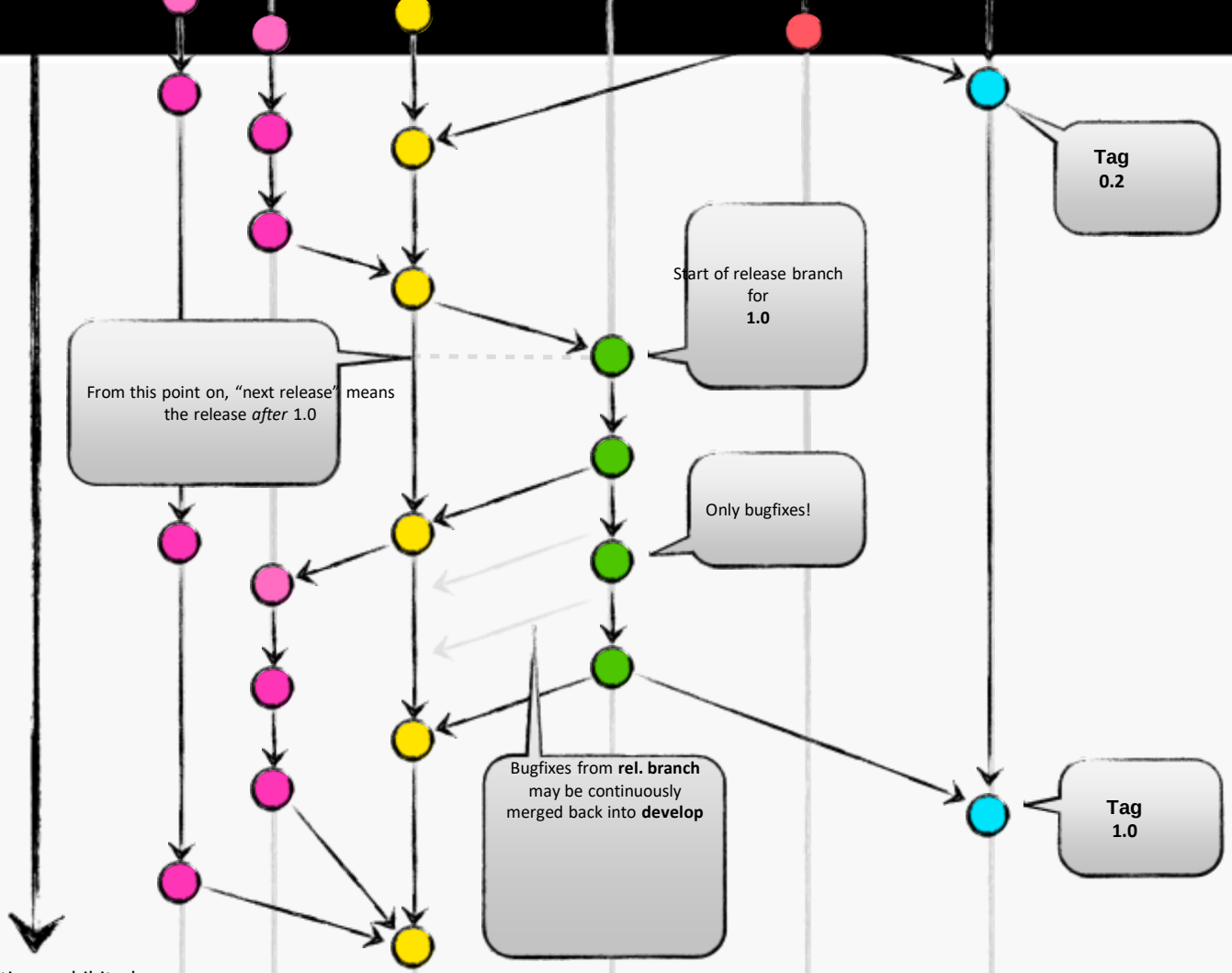
Git flow
A standard workflow for git

- Best practice
- Suitable for large teams large products
- Based on **two** main branches for development and releases



Example



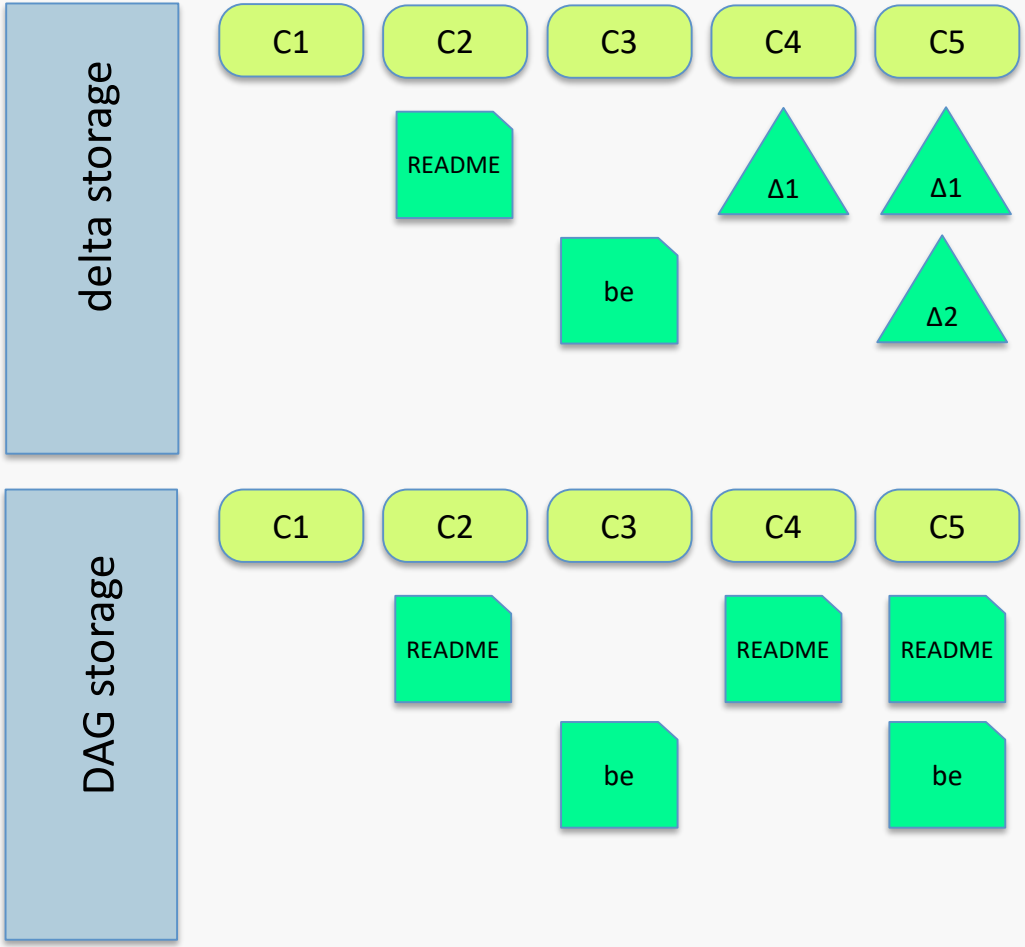


GIT is complex!

- GIT allows complex setups of repositories and workflow.
- Merge conflicts occur often and can be complex to be resolved.
- Repositories can and do get out of sync.
- **Problematic**: History can be changed in GIT!

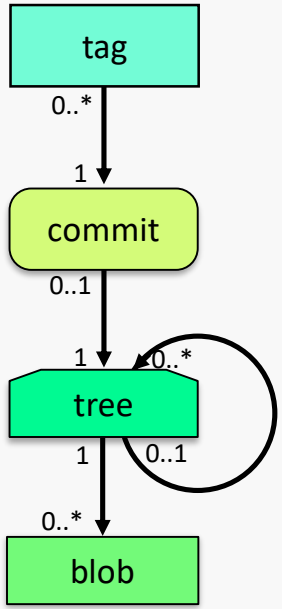
How does GIT work?

- Delta Storage vs Snapshots in DAG
- Objects in GIT
- Representation in the Repository



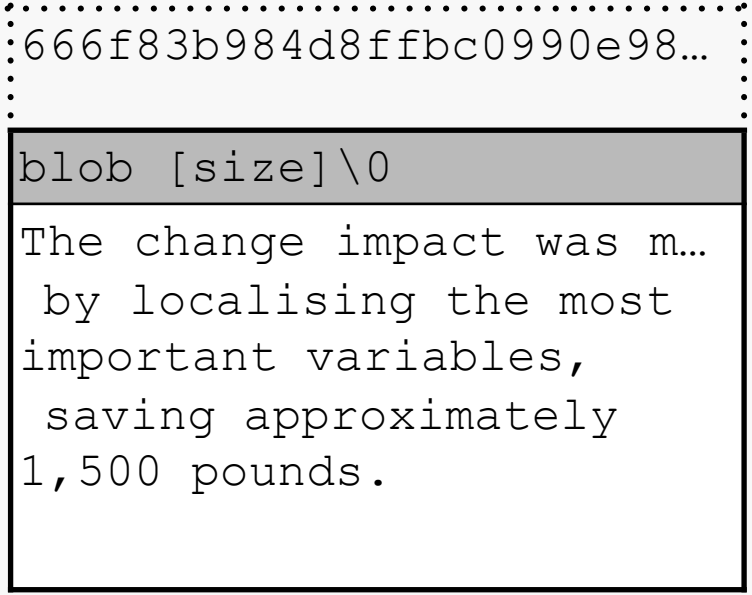
GIT Objects

- Blobs
- Trees
- Commits
- Tags



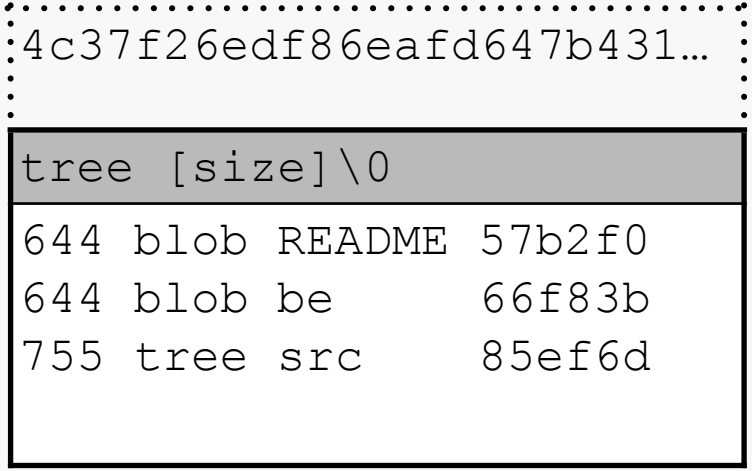
Blob

- Basic unit of GIT
- Contains the content (of a file)
- Referenced by the SHA1 of the content
- Immutable



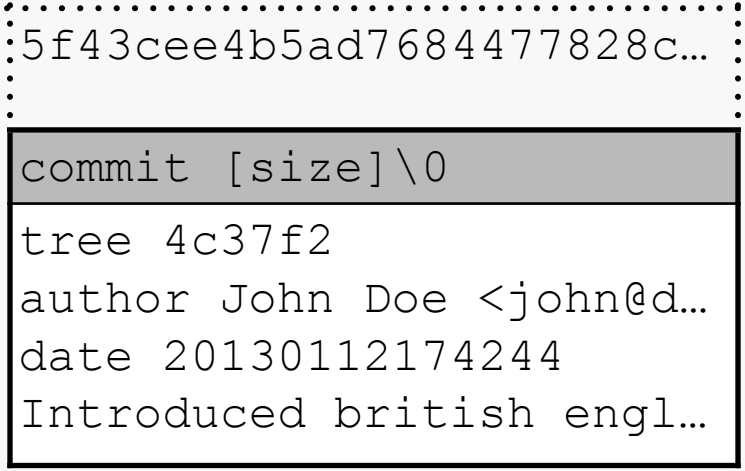
Tree

- A list of blobs and trees
- Contains filename and permission
- Is a blob itself



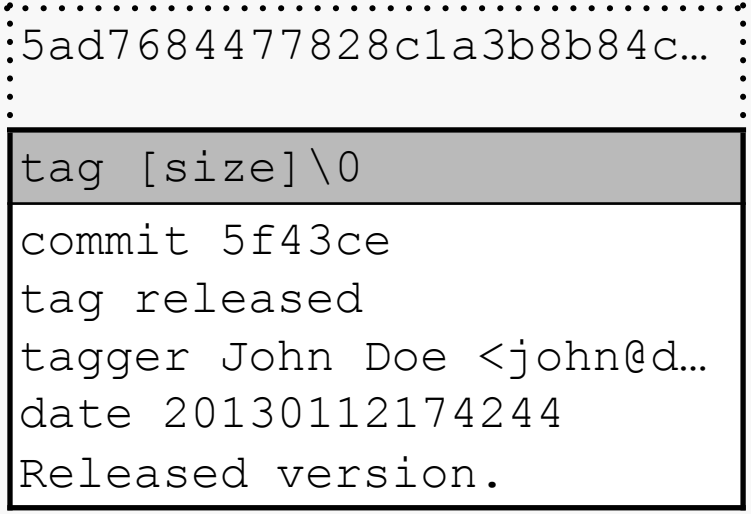
Commit

- References a rooted tree
- Contains information about the commit:
  - Author
  - Date
  - Message
- Is a blob itself

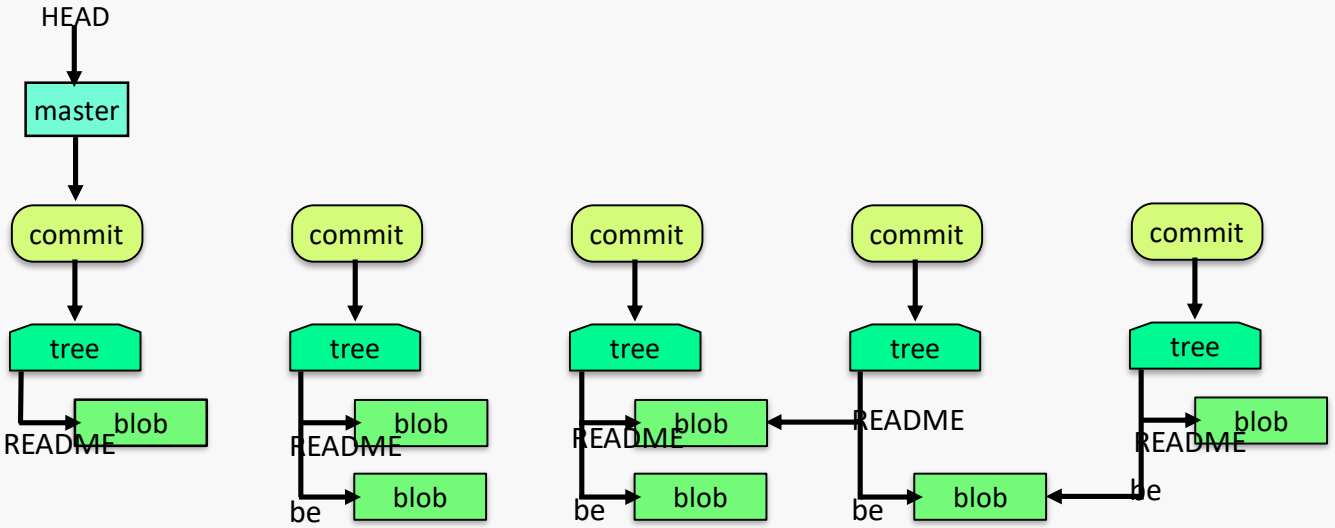


Tag

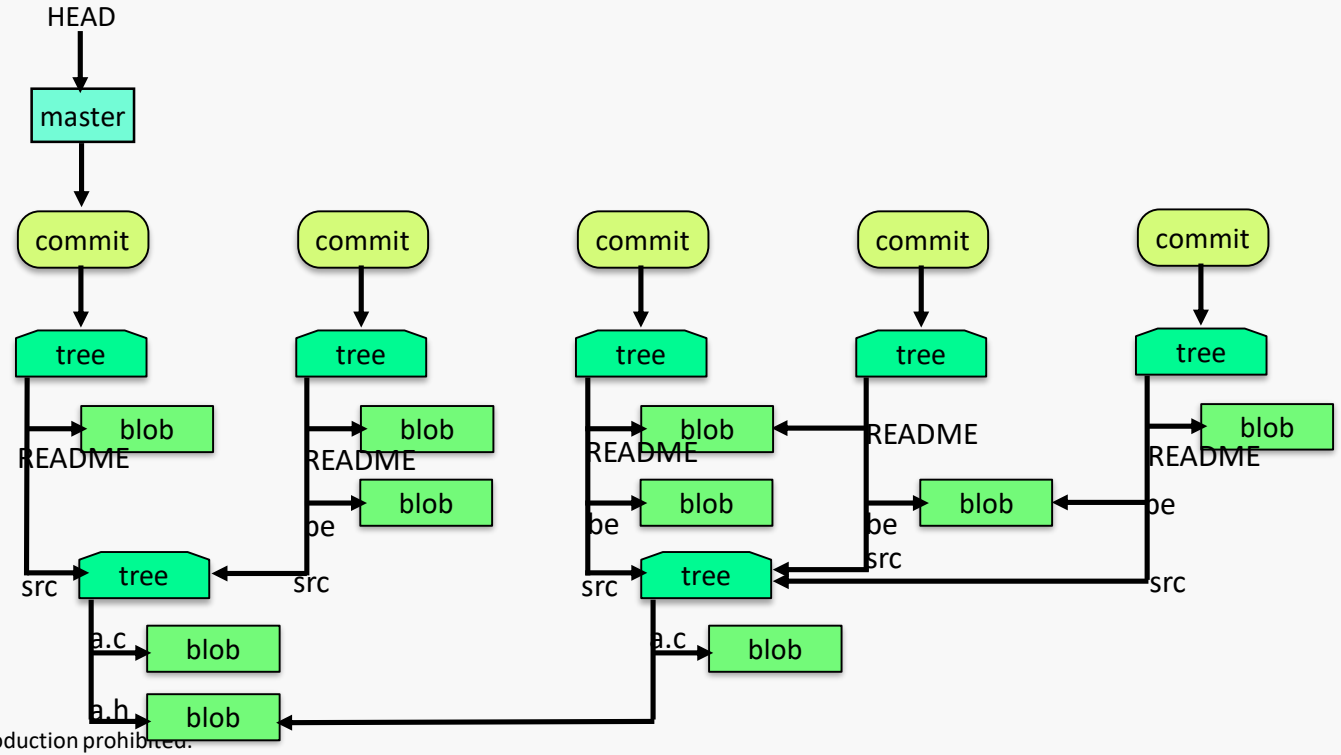
- A reference to a commit
- A file that contains the 40 SHA1 characters of the commit
- A commit can be referenced by any number of tags



The Repository
Example 1



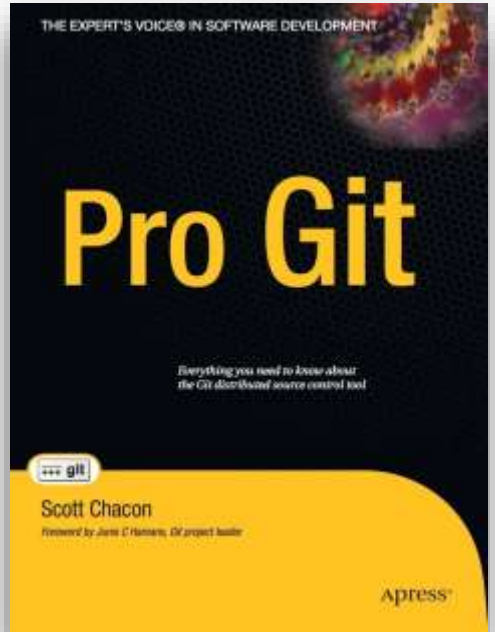
The Repository
Example 2



Homework

- Watch “Getting Git”
  by Scott Chacon
  http://vimeo.com/14629850
  (A one hour presentation
  about GIT)

- Read Chapter 2
  of “Pro Git”
  by Scott Chacon.



Concepts (1/2)

- GIT is a Distributed Version Control System
- The workspace has a local repository which can be synced with remote repositories.
- Repositories don’t have to be exact copies.
- Repositories are propagating changes by push, pull, or patches.

Concepts (2/2)

- GIT stores complete snapshots for commits, duplicates are omitted by references.
- References are generated through hashing the content (SHA1).
- GIT stores all data in four different objects: blob, tree, commit, tag.

Credits

- Git flow picture: Vincent Driessen
  http://nvie.com/posts/a-successful-git-branching-model/