Revision Control System

Understanding Revision Control System:

The case of Git

Dr. Juma Lungo

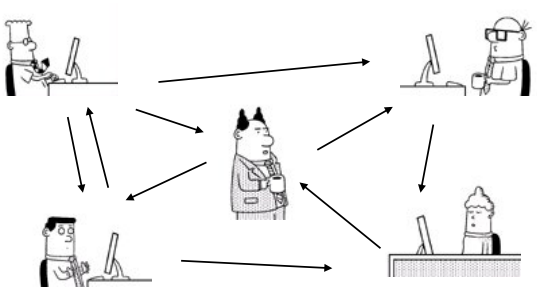
University of Dar-es-Salaam

Presentation Outline

- Problem Area
- Solution
- How it works
- Advantages
- Git commands
- Conflict resolutions

Problem area

- Software projects with multiple developers need to coordinate and synchronize the source code

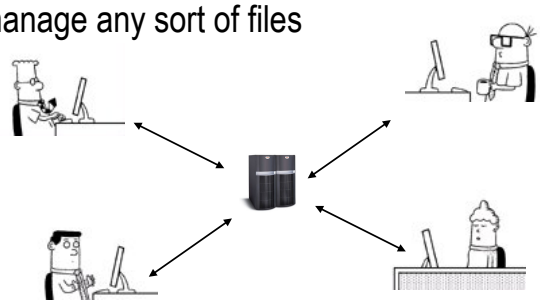


Approaches to version control

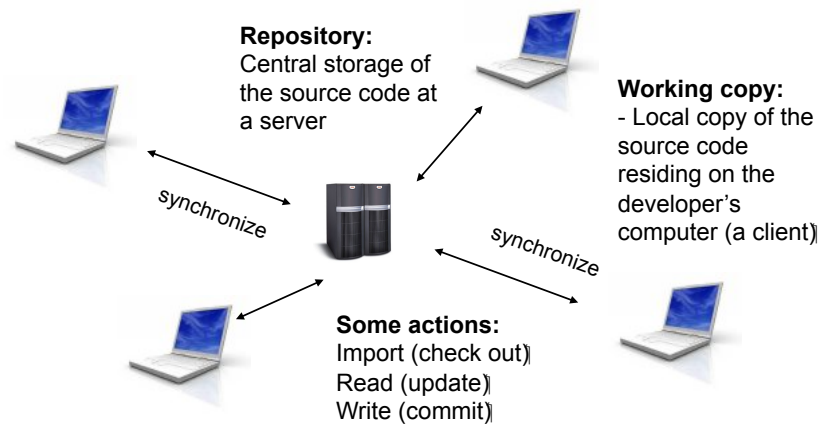
- Work on same computer and take turns coding
  - Nah...
- Send files by e-mail or put them online
  - Lots of manual work
- Put files on a shared disk
  - Files get overwritten or deleted and work is lost, lots of direct coordination
- In short: Error prone and inefficient

The preferred solution

- Use a revision control system (like Bazaar)
- RCS - software that allows for multiple developers to work on the same codebase in a coordinated fashion
- Can manage any sort of files

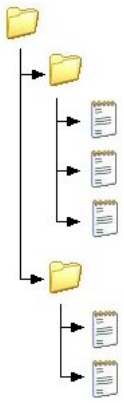


How it works



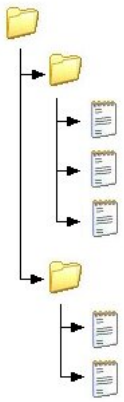
The repository

- A central store of data
- Stores information in a filesystem tree
- Remembers every change ever written to it
- Clients can check out an independent, private copy of the filesystem called a *working copy*
- Clients connect to the repository and read or write to the filesystem



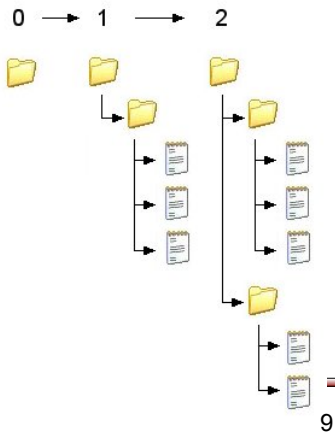
Working copies

- Ordinary directory tree
- Each directory contains an administrative directory
- Changes are not incorporated or published until you tell it to do so
- A working copy corresponds to a subtree of the repository



Revisions

- Every commit creates a new *revision*, which is identified by a unique revision number
- Every revision is remembered by the RCS and forms a revision history
- Every revision can be checked out independently
- The current revision can be rolled-back to any revision
- Commits are *atomic*



Work cycle

Initial check out:
The developer checks out the source code from the repository

1) Development:
The developer makes changes to the working copy

Client

2) Update:
The developer receives changes made by other developers and syncronizes his local working copy with the repository

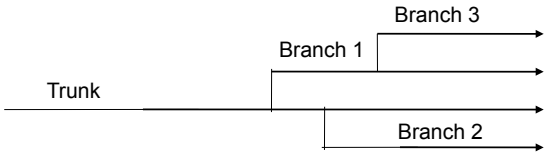
Repository

3) Resolve conflicts:
When a developer has made local changes that won't merge nicely with other changes, conflicts must be manually resolved

4) Commit:
The developer makes changes and writes or merges them back into the repository

Trunk and Branches

- Trunk is the original main line of development
- A branch is a copy of trunk which exists independently and is maintained separately
- Useful in several situations:
  - Large modifications which takes long time and affects other parts of the system
  - Different versions for production and development
  - Customised versions for different requirements



Advantages of RCS

- Concurrent development by multiple developers
- Possible to roll-back to earlier versions if development reaches a dead-end
- Allows for multiple versions (branches) of a system
- Logs useful for finding bugs
- Works as back-up

Good practises

- Update, build, test, *then* commit
  - Do not break the checked in copy
- Update out of habit before you start editing
  - Reduce your risk for integration problems
- Commit often
  - Reduce others risk for integration problems
- Check changes (diff) before committing
  - Don't commit unwanted code in the repo
- Do not use locking
  - Obstructs collaboration

What to add to the repository

- Source code including tests
- Resources like configuration files

- What to *not* add:
  - Compiled classes / binaries (the target folder)
  - IDE project files
  - Third party libraries

- Add sources, not products (generated files)!

Git Linux Commands

- Install Git
  - sudo apt install git

git config

- This command sets the author name and email address respectively to be used with your commits.
- Introduce yourself so that your work is properly identified in revision logs.
  - git config –global user.name “Juma Lungo”
  - git config –global user.email “jlungo@udsm.ac.tz”

git init

- This command is used to start a new repository.
- Usage:
  - git init staffcard
    - Where ”staffcard” is the ”repository name”

git clone

- This command is used to obtain a repository from an existing URL.
- Usage: git clone [url]
- Example:

  git clone git@github.com:jlungo/staffcard.git

git add – single file/directory

- This command adds a file to the staging area.
- Usage: git add [file]
- Example:

```
git add staff.php
```


git add – many files/directories

- This command adds one or more to the staging area.
- Usage: git add *
- Example:

```
git add *
```

git commit -m

- This command records or snapshots the file permanently in the version history.
- Usage: git commit -m “[Type in the commit message]”
- Example:

    git commit -m “updated staff form”

git commit -a

- This command commits any files you’ve added with the git add command and also commits any files you’ve changed since then.
- Usage: git commit -a
- Example:

    git commit -a

git diff

- This command shows the file differences which are not yet staged.
- Usage: git diff
- Example:

  git diff

git diff --staged

- This command shows the differences between the files in the staging area and the latest version present.
- Usage: git diff --staged
- Example:

  git diff --staged

git diff [first branch] [second branch]

- This command shows the differences between the two branches mentioned.
- Usage: git diff [first branch] [second branch]
- Example:

  git diff branch_1 branch_2

git reset

- This command unstages the file, but it preserves the file contents..
- Usage: git reset [file]
- Example:

  git reset staff.php

git reset [commit]

- This command undoes all the commits after the specified commit and preserves the changes locally.
- Usage: git reset [commit]
- Example:

  git reset 0jBXfD

  Where ”0jBXfD” is the commit hash tags

git reset --hard [commit]

- This command discards all history and goes back to the specified commit.
- Usage: git reset --hard [commit]
- Example:

  git reset --hard 0jBXfD

  Where ”0jBXfD” is the commit hash tags

git status

- This command lists all the files that have to be committed.
- Usage: git status
- Example:

```
git status
```


git rm

- This command deletes the file from your working directory and stages the deletion.
- Usage: git rm [file]
- Example:

```
git rm staff.php
```

git log

- This command is used to list the version history for the current branch.
- Usage: git log
- Example:

  git log

git log

```
edureka@master:~/Documents/DEMO$ git log
commit 09bb8e3f996eaf9a68ac5ba8d8b8fceb0e8641e7 (HEAD -> master)
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:25:17 2018 +0530

    Changes made in HTML and CSS file

commit b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:13:29 2018 +0530

    CHanges made in HTML file

commit aff3269a856ed251bfdf7ef87acb1716a2a9527a
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:07:28 2018 +0530

    First Commit
```

git log --follow[file]

- This command lists version history for a file, including the renaming of files also.
- Usage: git log –follow[file]
- Example:

  git log –follow project_1

git log

```
edureka@master:~/Documents/DEMO$ git log --follow project_1
commit 2b4c50431c127a0ae9ede4aace0b8dd1f9fcf2c5
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:50:08 2018 +0530

    New file added

commit 09bb8e3f996eaf9a68ac5ba8d8b8fceb0e8641e7
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:25:17 2018 +0530

    Changes made in HTML and CSS file

commit b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:13:29 2018 +0530

    CHanges made in HTML file
```

git show

- This command shows the metadata and content changes of the specified commit.
- Usage: git show [commit]
- Example:

  git show 0jBXfD

  Where ”0jBXfD” is the commit hash tags

git tag

This command is used to give tags to the specified commit.

- Usage: git tag [commitID]
- Example:

  git tag 0jBXfD

  Where ”0jBXfD” is the commit hash tags

git branch

- This command lists all the local branches in the current repository
- Usage: git branch

Example:

```
git branch
```


git branch [branch name]

- This command creates a new branch.
- Usage: git branch [branch name]
- Example:

```
git branch branch_1
```

git branch -d [branch name]

- This command deletes the feature branch.
- Usage: git branch -d [branch name]
- Example:

  git branch -d branch_1

git checkout

- This command is used to switch from one branch to another.
- Usage: git checkout [branch name]
- Example:

  git checkout branch_2

git checkout -b [branch name]

- This command creates a new branch and also switches to it.
- Usage: git checkout -b [branch name]
- Example:

  git checkout -b branch_3

git merge

- This command merges the specified branch's history into the current branch.
- Usage: git merge [branch name]
- Example:

  git merge branch_2

git remote add [variable name] [Remote Server Link]

- This command is used to connect your local repository to the remote server.
- Usage: git remote add [variable name] [Remote Server Link]
- Example:

  git remote add git@github.com:jlungo/staffcard.git

git push [variable name] master

- This command merges the specified branch's history into the current branch.
- Usage: git push [variable name] master
- Example:

  git push origin master

git push [variable name] [branch]

- This command sends the branch commits to your remote repository.
- Usage: git push [variable name] [branch]
- Example:

  git push origin master

git push –all [variable name]

- This command pushes all branches to your remote repository.
- Usage: git push –all [variable name]
- Example:

  git push –all origin

git push [variable name] :[branch name]

- This command deletes a branch on your remote repository.
- Usage: git push [variable name] : [branch name]
- Example:

  git push origin : branch_2

git pull

- This command fetches and merges changes on the remote server to your working directory.
- Usage: git pull [Repository Link]
- Example:

  git pull git@github.com:jlungo/staffcard.git

git stash save

- This command temporarily stores all the modified tracked files.
- Usage: git stash save
- Example:

```
git stash save
```


git stash pop

- This command restores the most recently stashed files.
- Usage: git stash pop
- Example:

```
git stash pop
```

git stash list

- This command lists all stashed changesets.
- Usage: git stash list
- Example:

  git stash list

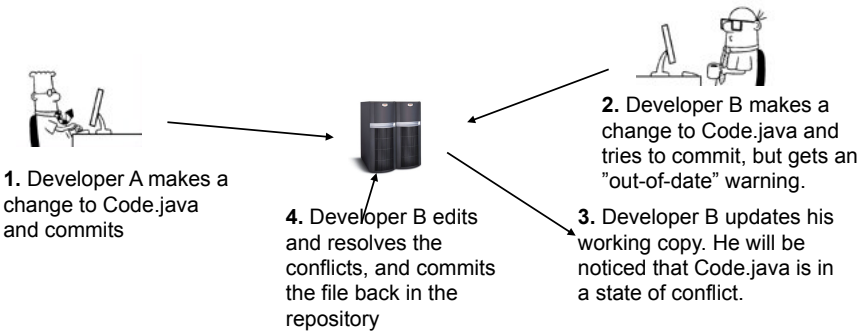
git stash drop [stash id]

- This command discards the most recently stashed changeset.
- Usage: git stash drop [stash id]
- Example:

  git stash drop stash@(0)

Conflicts

- Arises if several developers edit the same part of a file
- Solution in Subversion: ”Copy-modify-merge”



Conflicts

- Changes that do not overlap are merged automatically
- 4 solutions are provided in conflict situations:
  - Use ”mine” version – the developers local copy
  - Use ”their” version – the copy in the repository
  - Use ”base” version – the file before you started editing
  - Use the original file with conflict markers and edit the conflict manually before comitting
- Subversion must be told that the conflict is *resolved*
  - Will remove the temporary files and let you commit

Resolving a merge conflict (Page 1)

- Merge conflicts occur when competing changes are made to the same line of a file, or when one person edits a file and another person deletes the same file.
- To resolve a merge conflict caused by competing line changes, you must choose which changes to incorporate from the different branches in a new commit.

Resolving a merge conflict (Page 2)

- For example,
  - If you and another person both edited the file "*index.php*" on the same lines in different branches of the same Git repository, you'll get a merge conflict error when you try to merge these branches.

Resolving a merge conflict (Page 3)

- Open Terminal
  - Ctrl + Alt +T
- Navigate into the local Git repository that has the merge conflict.
  - cd staffcard
- Generate a list of the files affected by the merge conflict.
  - git status

Resolving a merge conflict (Page 4)

```
$ git status
> # On branch branch-b
> # You have unmerged paths.
> #   (fix conflicts and run "git commit")
> #
> # Unmerged paths:
> #   (use "git add ..." to mark resolution)
> #
> # both modified:      styleguide.md
> #
> no changes added to commit (use "git add" and/or "git commit -a")
```

Resolving a merge conflict (Page 5)

- Open your favorite text editor
- When you open the file in your text editor, you'll see the changes from the HEAD or base branch after the line <<<<<<< HEAD.
- Next, you'll see =======, which divides your changes from the changes in the other branch, followed by >>>>>>> BRANCH-NAME.

Resolving a merge conflict (Page 6)

- In this example, one person wrote
  - "open an issue" in the base or HEAD branch
- Another person wrote
  - "ask your question in IRC" in the compare branch or branch-a.

Resolving a merge conflict (Page 7)

```
If you have questions, please
<<<<<<< HEAD
open an issue
=======
ask your question in IRC.
>>>>>>> branch-a
```


Resolving a merge conflict (Page 8)

- Decide if you want to keep only your branch's changes, keep only the other branch's changes, or make a brand new change, which may incorporate changes from both branches.
- Delete the conflict markers <<<<<<<, =======, >>>>>>> and make the changes you want in the final merge.

Resolving a merge conflict (Page 9)

- In this example, both changes are incorporated into the final merge:
- ***If you have questions, please open an issue or ask in our IRC channel if it's more urgent.***
- Add or stage your changes.
  - **git add .**
- Commit your changes with a comment.
  - **git commit -m "Resolved merge conflict".**

The end!

Q & A