Introduction to VCS

What is a VCS ?

- Version Control System == VCS
  - A system to archive and retrieve multiple versions of a set of file (usually computer sources)

Advantages of VCS

- Is it always possible to retrieve a previous version.
  - If a version is broken, you can always get the previous one
  - Why not use backups?
    - Which is older and which is newer?
    - How to find a non-broken version?

Advantages of VCS

- It is possible to share work and collaborate on it.
  - Without requiring access to the same workstation
  - Allowing (most) of the work to be unsynchronized

Basic Concepts - Repository

- A Repository is the archive of all versions of all the files of a project.
- It allows registering new versions and retrieving new or old ones.
- It allows to reconstruct the state of the project at any specific point in time

Basic Concepts - Commit

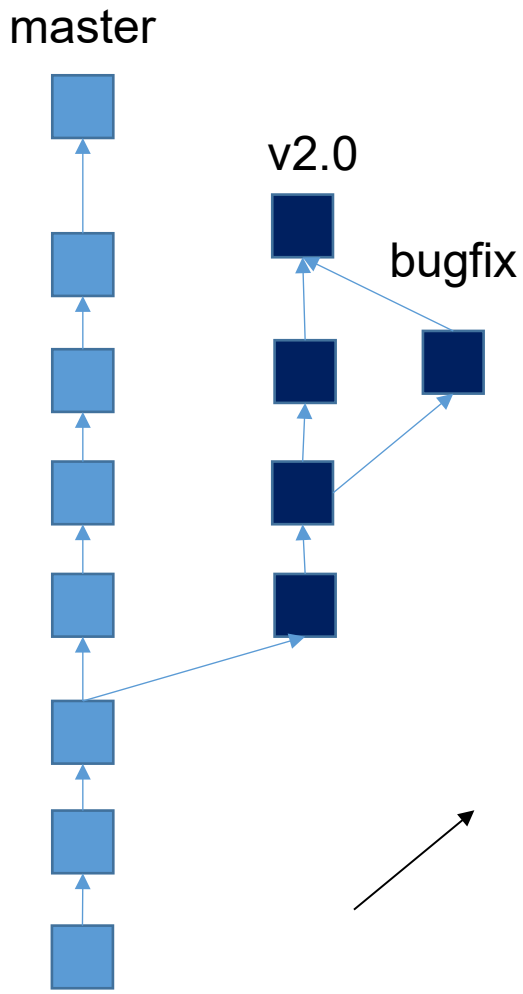
- A commit is the act a registering a new file / set of files (or new versions of them) in the repository
- In modern VCS, the operation is atomic
- Once successfully registered, even if your version is lost, it can be retrieved from the repository.

Basic Concepts - Branch

- A VCS can have multiple concurrent histories for the same files
  - It maps to multiple concurrent lines of development.
- Each such history is called a “branch”
- Usually there is a “main” branch called “master” or “trunk” or “main” depending on the VCS.

Basic Concepts - Branch

- Each branch may have different version of each file.
- It is possible to choose on which branch to work.
- It is possible to take all changes (commits) from one branch and copy them to a different branch (merge).



Basic Concepts – Merge

- A merge is the operation of taking the changes (commits) on a branch and applying them on a different branch.
- It is used to reconcile different lines of development.
- It is possible to have multiple conflicting changes on the same file (conflict) which must be resolved.
  - Examples will be given during the git-specific part.

Basic Concepts – Tag

- A tag is a human-readable name associated to a specific version of the project.
  - Most often used to identify a release
    - v1.3.4, project_3.4.5, etc…
  - But it can be everything

GIT Basics

Why GIT?

- Well, let's see…
- Effortless branching AND merging
  - Both direct and reverse
  - Keeps track of multiple merges
    - Much less spurious conflicts
  - Supports tools for three way merges
    - Very useful for conflict resolution!
  - Allows partial merging
    - Merge only THIS commit from branch A to branch B
      - Also known as: backport this feature from release A to release B
  - Local branches
    - Only on your dev machines
- Branches are cheap
  - Create a new branch -> Create a new file containing one line.

Why GIT?

- Fully local commands
  - All commands (commit included) work locally and do not require networking
- Debug support!
  - Find the exact commit that introduced a bug!
- “Floating” commits
  - I.e: keep track of the changes I made, but do not commit them on any branch
- Easy reverts
  - Not just on local copy, but also on committed changes – even many releases before
- Full history
  - Always have the full history of every file available locally

Why GIT

- Distributed repositories
  - Each has his own local copy
  - Synchronization via push/pull/patches

Why GIT

- Exotic features
  - Submodule -> establish a link from your module to any other GIT repository
  - Multiple remotes -> Download changes from multiple remote repositories
  - Rebase
  - Rerere

Basic Git w.r.t basic CVS/SVN

- cvs/svn commit
- cvs/svn add
- cvs/svn tag
- cvs/svn branch
- cvs/svn co – svn revert
- cvs/svn merge
- cvs/svn log
- cvs/svn status
- svn revert

- git commit
- git add
- git tag
- git branch
- git checkout
- git merge
- git log
- git status
- git revert

Concept 1: The Index

- The Index is an intermediate step between the local copy and the repository
  - Things are first added to the Index and only then committed.
  - Only things added to the Index can be committed in the repository.

Concept 1: The Index

- Why an Index?
  - It avoids accidental commits, i.e: committing more than was intended.
- How to see its contents?
  - git status
  - In the following: staged == added to index

Concept 2: Commit and commit names

- The basic information tracked by git is the single commit.
  - Cf: CVS -> The file
  - Cf: SVN -> The project
- All commands that somehow refer to a specific state of the repository refer to a specific commit.
  - E.g: commit, tag, push, pull, merge, reset, etc…

Concept 2: Commits and commit names

- How to refer to a commit:
  - Its absolute name: d6a7c255a85eaa1e8148d35187f9d32bb63d13f7
    - Can be abbreviated: d6a7c255a8
  - Top commit on a branch: HEAD, <branch name>
    - Full symbolic name: refs/heads/<branch name>
  - Tag name: <tag name>
    - Full symbolic name: refs/tags/<tag name>
  - Relative commit: HEAD^ (penultimate), HEAD^^ , HEAD~2
- In all cases, omitting the name means HEAD

Concept 2: Commits and commit names

- Two special commit names have only limited validity:
  - ORIG_HEAD: The head of the current branch BEFORE the last merge. Only available after a git merge
  - FETCH_HEAD: The head of the currently fetched branch. Only available after a git fetch.

Concept 2: Commits and commit names

- Every commit may have any number of fathers and any number of children
  - Fathers -> all commits that precede it in some branch
  - Children -> all commits that follow it in some branch
- The name of the branch is a synonym for the commit at the top of that branch
- Every command that accepts a branch name accepts a commit name in its place

Local and remote commands

- Most commands work on the copy of the repo you have on your development machine
  - This includes *commit*
- Only a select few require network connectivity
  - In this lesson: *clone, push, pull, fetch*

Initial configuration

- git config --global user.name “Pinco Pallino”
- git config --global user.email pinco.pallino@example.org
- git config --global push.default simple

- These commands setup information that will be included in all your commits
- Git will tell about them if you forget to use them

clone

- git clone <repo>
  - Downloads on your machine a FULL copy of the repository
    - All the branches, all the tags, all the history from the beginning of the project
  - Most other commands will NOT need network connectivity.
  - For this lesson: git clone git@baltig.infn.it:vciaschini/corsogit.git

Local usage

- Commands and features when working on the local repository
  - help, add, rm, mv, status, checkout, branch, commit, diff, log, tag, merge, reset, revert, stash

help

- git help <command name>
  - Prints the man page for the specific git command
  - Ex: git help checkout

status

- git status
  - Shows the status of the working copy
  - What is modified
  - What has been staged
  - How to unmodify
  - How to unstage

add, rm

- Add a new file or files to the repository
  - git add <file> … <file>
  - git commit
- Remove a file:
  - git rm <file> … <file.
  - git commit

mv

- Copies a file: two way:
  - git mv <old> <new>
  - git commit
- Second way
  - mv <old> <new>
  - git rm <old>
  - git add <new>
  - git commit
- Note that git can auto-detect moving files.

checkout

- Basic usage:
  - git checkout <branch>
  - git checkout <tag>
  - Note that for this command there is no difference between branch and tag
    - You can also commit over a tag. But the tag will point to the old version
  - Further usages of checkout will be explained later

Commits

- To commit, you must first stage
  - git add <files>
  - git commit
  - Exactly the same as adding a completely new file
- Or you can say “stage *and* commit all known files.”
  - git commit –a
  - Still new files must be staged explicitly

diff

- To show what has changed in the code:
  - git diff
- Common Syntaxes
  - git diff
    - Shows changes relative to the index
  - git diff –cached
    - Shows changes between the index and the last commit
  - git diff <commit>..<commit>
    - Shows changes between the two commits
  - git diff <commit>
    - Shows changes between working dir and commit

Branches

- There are two kinds of branches: local and remote
  - Local branches are only on your repository
  - Remote branches are on remote repositories
- Do not do commits on top of remote branches. They will eventually be lost. (git will warn of this)

Branches

- What local branches do exist?
  - git branch
- What remote branches do exist?
  - git branch –r
- What branches exist? (local and remote)
  - git branch –a

Branches

- Create a branch of an existing one
  - git branch <new branch> <oldbranch>
- If <oldbranch> is omitted, it is the branch currently checked out.
- Special case if <oldbranch> is a remote one.
  - <new branch> becomes a tracking branch
    - Everything working on non-tracking branches works on tracking ones.
    - Additional properties will be explained when dealing with remote repositories.

Tags

- Adds a symbolic name
  - git tag <tag name> <commit name>
- A tag cannot be changed. However, it can be moved (See the advanced session for instructions)

Merges

- Basic usage:
  - git merge <branch name>
  - Merges <branch name> in the current branch
  - Note that a successful merge implies an automatic commit
    - Add –no-commit if you do not want it.

Merge variants

- git merge <branch 1> … <branch n>
  - Attempts merging multiple branches at the same time.
  - Fails in case of conflicts
- Note that criss-cross merges are considered “normal” and require no special handling

Merges: Resolving conflicts

- Done the usual way, or:
- Can use external tools
  - Kdiff3 example
- Then do the usual ‘add & commit’

- Note that in general git is a lot smarter than svn or cvs when resolving conflicts automatically.

reverts:

- Three main cases:
  - Is the file only on your working copy?
    - git checkout -- <file>
  - Is the file staged?
    - git reset HEAD <file>
  - Is this a commit?
    - git revert <commit>
    - This one will put on the repository a new commit that inverts the one above.

reset

- What is reset?
  - Reset throws away some changes.
  - Main usages:
    - git reset –hard
      - Reverts your local copy to the latest committed one and unstages everything
    - git reset –hard <commit>
      - Reverts your local copy to <commit> and unstages everything.
      - This also removes all commits done after <commit> from the repository

log

- Show the commits on the current checkout
  - git log
- Annotate them with tag name
  - git log –decorate
- Personally, I prefer using gitk

stash

- There can be multiple stashes, each with a name.
  - git stash list
- You can apply a stash other than the last one.
  - git stash apply <stash name>
- Applying a stash does not delete it.
  - git stash drop <stash name>

Working with remote repositories

- Commands for synching with remote repositories:
  - clone, push, fetch, pull

push

- Ok, you have done your work. Now you want to make it available to the other developers. Use git push.
- git push
  - If there are tracking branches, pushes commits from those to the remote ones
  - Non tracking branches are ignored
  - Pushes only the branch currently checked out.
  - git push –all pushes all branches

push

- But if I want to push a new branch?
  - git push –u origin <new branch>
    - This also creates a remote branch on the local repository along with the tracking relation
- It is also possible to delete a remote branch
  - git push –delete <remote branch>
- Tags are not pushed by default
  - git push –tags
  - Also pushes all tags on tracking branches
  - git push –tags <tag>
  - Only pushes that tag

fetch

- Fetch allows to receive commits from remote repositories
  - git fetch
    - Fetches the remote version of the current branch
    - Only for tracking branches – does not touch the corresponding local branch or the working copy.
  - Use: git merge origin/master to merge

pull

- Does git fetch + git merge the remote changes in the local branch.
  - This command WILL change your local checkout

Additional tips and tricks

- Did I merge branch X anywhere? Where?
  - git branch –contains <X>
    - Prints all the branches that contain a fully merged branch X
- How do I delete branch X ?
  - git branch –d <X>
    - Only works if it is fully merged in at least another branch
  - git branch –force –d <X>
    - Deletes unconditionally

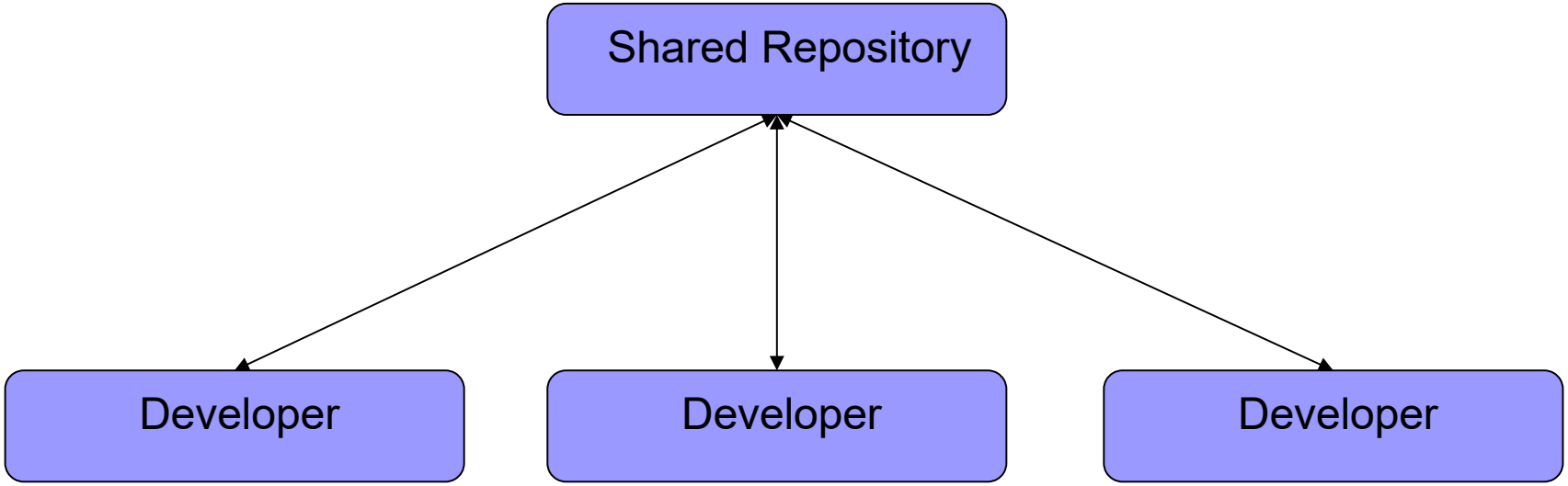
Additional tips and tricks

- Gitk
  - The repository visualizer
    - Very useful for keeping track of merges
    - I keep it perpetually open

Creating a new repository

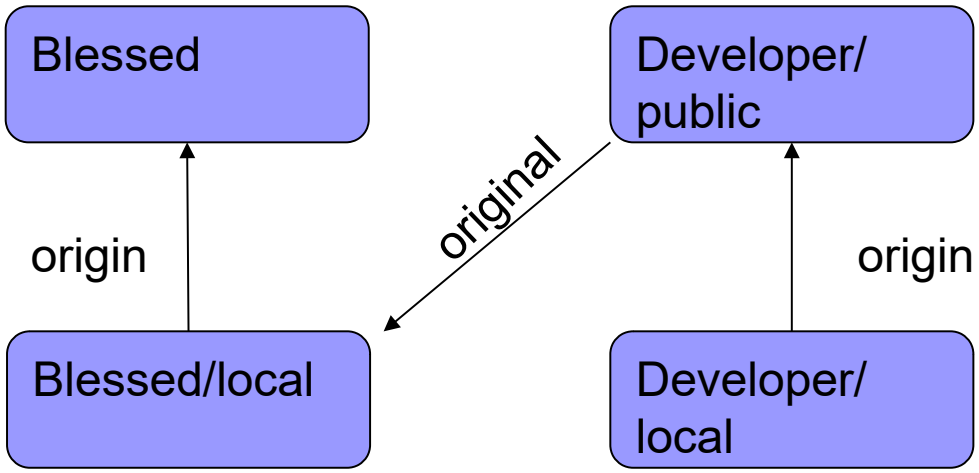
- Inside the directory:
  - Remove all files that should not be committed
  - git init
  - git add .
  - git commit
- Now you have everything, but you cannot use it as an unattended remote
  - git add remote origin <path>
  - git push

Some typical workflows



- Many developers, central “blessed” repository

One Blessed Repo



- One integration manager

Which workflow?

- Project's choice
- Most of this lesson is accurate for both choices.
  - Actually, in many cases there is no difference in day-to-day use.
- Specific issues for the second workflow will be explained in dedicated slides