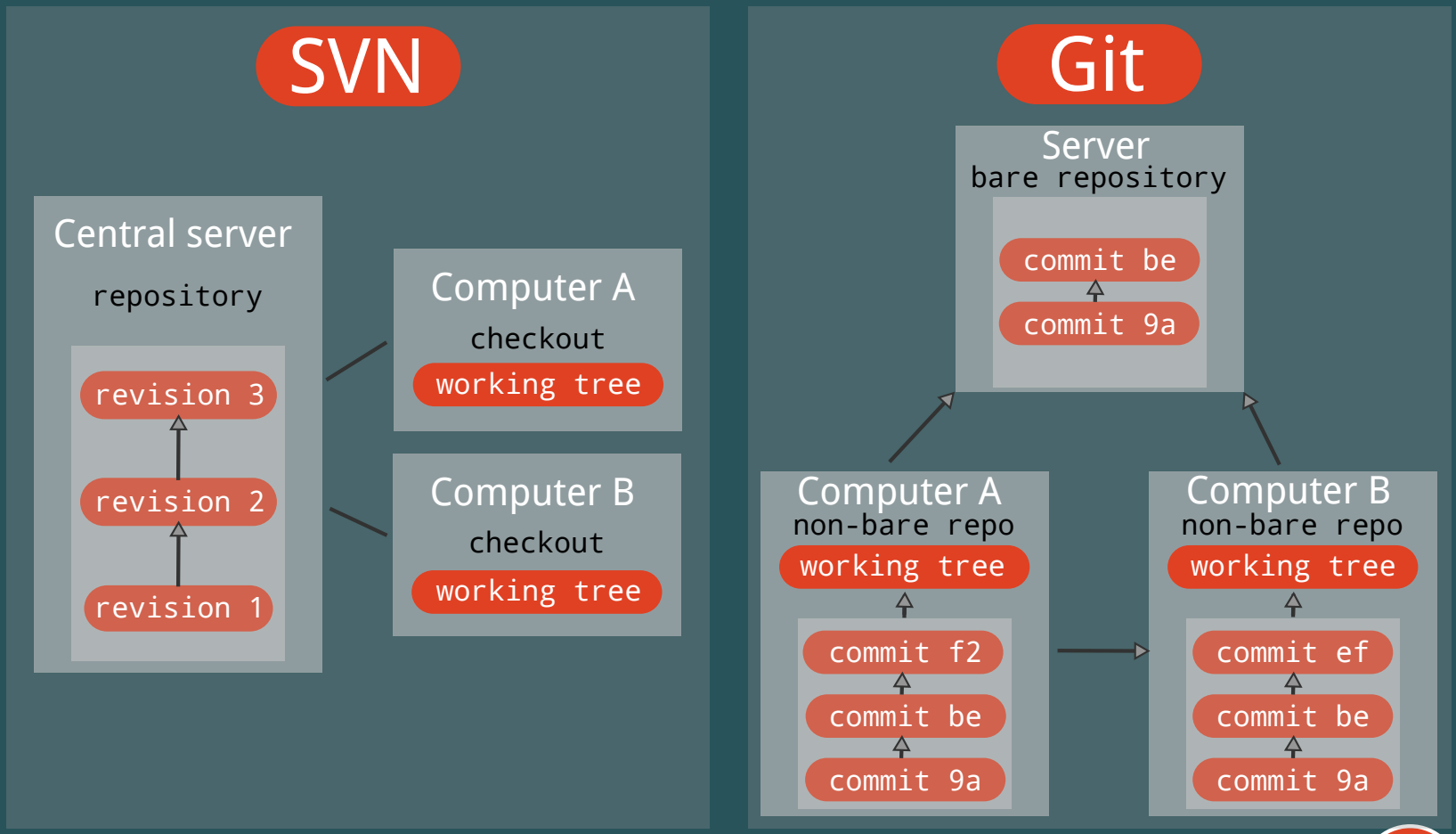
GIT Tutorial

A quick overview

Guillaume Seguin
Jean-Baptiste Alayrac

SVN vs Git

two different principles...



SVN vs Git
main differences...

SVN

easy to use

slower

needs connection

works offline

faster

super safe

branches are easier

Git

learning curve

Git prerequisite
before starting...

Set up your ssh keys

Create one key once and for all

Command line : `ssh keygen -t rsa`

This command creates two files in `~/.ssh` : `id_rsa` and `id_rsa.pub`

Then add `id_rsa.pub` to your Inria forge account.

NB : also useful for connecting to lab assets with `ssh`.

About graphical user interface

Command line tutorial

However we give some good tools if you are command line allergic.

Linux : gitg, giggle.

Mac : SourceTree

Windows : SourceTree, TortoiseGit

NB : the GUI are particularly useful to display your repo as a nice tree and also allows to display nicer diff of files.

Creating a new repository

Using Inria Forge tool

Secure and convenient for Inria people

- Create a new project, wait for its approval
- `Project > Tool` : check use source code
- `SCM > Administration` : Choose Git
- `SCM > View Code Source` : Check the repo is fine

NB : You can also create a repository directly on your machine.
See the command kit at the end for that.

Clone the repository

copy everything on your machine

`git clone "url"`

copy the repository that is at "url" to your machine

This url can be found on Inria Forge.

Equivalent on SVN : `svn checkout "url"`

NB : a folder `.git` is created in your working space.

Inspect the repository

tools to track the project...

`git log`

Show commit logs

`git show "commit-id"`

Shows the log message and textual diff of the commit

`git status`

Show the working tree status

Add files to track

Let git know about the interesting things

`git add`

Add file contents ("stage" them) to the index

Index : list of things to store in the next commit.

Super important : create a .gitignore file to specify files to never store, e.g. latex temporary file.

For a LaTeX paper :

```
*.aux
*.bbl
*.blg
*.brf
*.log
*.swp
*.swo
*~
papername.pdf
```

Commit your changes

Do it very regularly, it is local so super fast!

`git commit`

Record changes to the repository

`git commit -a` : any change to files that have ever been added to the repositiory will be stored

`git commit FILE1 FILE2` : commit changes to FILE1 and FILE2

`git commit` : any change that has been added to the index will be stored (use `git add`)

Push your changes

Synchronize with the remote repository

`git push`

Show commit logs

Two things can happen :
1) It works. All good !
2) `![rejected] master -> master (non-fast-forward)` : there were new changes to the repository, you have to get them first

`git commit + git push`

=

`svn commit`

Pull

Get updates from the remote repository

`git pull`

Fetch and integrate changes from the remote repo

If you have made commits since the last pull, Git will either:
1) Successfully merge the two trees. A merge commit is created automatically.
2) Automatic merging fails, you have to fix the conflicts yourself.

Equivalent on SVN:

`svn update`

Pull
Solving conflicts

Git will edit the problematic files to show the conflicting parts:

```
<<<<<<< HEAD (this is your repository)
    UberPOP rocks !
=======
    Taxis are so cool !
>>>>>>> master (this is the remote repository)
```

Locate these blocks, replace them with the correct version,
and mark that the files have been fixed using `git add FILENAMES`
Once all conflicts are fixed, use `git commit` to finalize.

Use `git rm FILENAMES` if you simply want to remove the files, e.g. for conflicts in .bbl files added by mistake)

Coming back to a previous commit

Solving conflicts

`git checkout commit-id`

Checkout a given commit to the working tree

`git reset --hard commit-id`

Hard reset everything to the given commit

Do it only if you know what you're doing ! (especially if you have pushed already)

NB : a safest way to use `git reset --hard` is to first create a branch on which you will let the repo as is. On the previous branch you can then do the reset. This way you will still be able to recover the commits which were made after the reset by simply going to your new branch.

Mark a special commit

`git tag tagname [commit-id]`

Tag current commit (or given commit) with tagname

e.g. `paper-submitted`
`paper-arxiv`
`paper-rebuttal`
`paper-final`

Use `git push --tags` to push tags

Take home commands

```
git clone = svn checkout
git pull = svn up
git add = svn add
git commit + git push = svn commit
git mv = svn mv
git rm = svn rm
git status = svn status
git diff : show currently uncommited & unadded changes
git reset --hard = svn revert
git checkout "commit-id" = svn update -rXXX
git reset --hard "commit-id" : revert to a given commit /!\
git tag : tag a commit to mark an important revision
git add -i : interactively add some chunks to the index
git stash : temporarily remove your uncommited changes
git stash apply : reapply previously stashed changes
mkdir proj && cd proj && git init : create new local repo
git branch "branchname" : create branch from current commit
```