Introduction to

Samuel Iglesias Gonsálvez
siglesia@cern.ch
BE/CO/HT
Index

• Introduction to Control Version systems.
  • Centralized systems (e.g. CVS, SVN).
  • Distributed systems (e.g. GIT, Mercurial, Bazaar).
  • Branch.

• Basics:
  • Create a repo, clone, commit, merge, etc.

• Best practices:
  • Tips, recommendations, suggestions.
What is a Version Control System?

- Track changes to the source code in a systematic way.
- Maintain a history of changes
  - Allows to rollback changes.
  - Log of changes. You know who to blame :-)
  - Kind of redundancy, backup... whatever.
  - Everyone can see each commit and see the changes.
Centralized systems

- One server with all the data.
- Lots of clients.
  - When the user does a commit, it goes to the server.
  - The use of branches for development is not widespread.
- Very easy to learn and to understand.
- What happens in case of problems?
  - Merges, conflicts on files.
  - If you don't have connection to the server, you cannot commit anything → bad, bad, bad...
  - Single point of failure.
Centralized systems
Distributed system

- Anyone can be a server:
  - Multiple points of failure $\rightarrow$ more reliability.
  - The local repo can be a server $\rightarrow$ commit without connection to Internet.
- Easy to create, merge, delete branches.
- More complex.
Distributed system
Branches
Feature for future release

Major feature for next release

Incorporate bugfix in develop

Severe bug fixed for production: hotfix 0.2

Tag 0.2

From this point on, "next release" means the release after 1.0

Start of release branch for 1.0

Only bugfixes!

Bugfixes from rel. branch may be continuously merged back into develop

Tag 1.0
“Old” workflow

Remote Server → checkout → Code Snippets

Code Snippets:
- Code 1
- Code 2
- Code 3
“Old” workflow
“New” workflow

Remote Server → clone → code snippets
“New” workflow

Remote Server

Local repo

commit
“New” workflow

Remote Server

commit

Local repo
“New” workflow
“New” workflow

Remote Server → Push → Local repo

Commit 1
Commit 2
Commit 3
“New” workflow

Remote Server

Commit 1
Commit 2
Commit 3

Push

Local repo
Basics

• Create a new repo:
  
  `git init`

• Basic configuration:
  
  `git config --global user.name "John Doe"`
  `git config --global user.email john.doe@cern.ch`

• Clone a repository:
  
  `git clone <url> <name>`
Basics

- Permanent configuration → Use of .gitconfig

test@test:~$ cat $HOME/.gitconfig

[ user ]

  name = John Doe
  email = john.doe@cern.ch

[ core ]

  editor = vim
Basics

- Add a remote repository:
  
  ```bash
  git remote add <name> <url>
  ```

- Fetch data from the remote repository:
  
  ```bash
  git fetch <name>
  ```

- Pick all the committed files from the remote repo and put them in the active branch:
  
  ```bash
  git pull <name> -u <branch>
  ```
Basics

• Add a file:
  
  ```
  git add <name_file>
  ```

• Move a file:
  
  ```
  git mv <file>
  ```

• Delete a file:
  
  ```
  git rm <file>
  ```

• Commit:
  
  ```
  git commit <files>
  ```
Basics

• Show the history:
  `git log [<file/dir>]`
  `git log --pretty=oneline`

• Show differences the actual state and an older commit:
  `git diff <commit>`
Basics

• See all branches:
  
  `git branch -a`

• Create a new branch:
  
  `git checkout -b <name_branch>`

• Change to another branch:
  
  `git checkout <name_branch>`

• Delete a branch:
  
  `git branch -D <name_branch>`
Basics

• Merge a branch into the active one:
  
git merge <name_branch>

• Import changes from another branch (remote or local) and put the active branch ones on top:
  
git rebase <name_branch>

• Edit the commit message from the latest commit:
  
git commit --amend
Basics

• Help:
  
git help
  
git help <topic>
  
• Documentation:
  
man git
  
http://book.git-scm.com/
  
http://progit.org/book/
  
http://gitimmersion.com/
  
http://www-cs-students.stanford.edu/~blynn/gitmagic/
Best practices

- Commit small changes, commit often.
  - Don't use it as a backup.
  - Commit stable changes, i.e., it compiles and doesn't break the tree.
- Commit message clear and understandable.
  - One line to resume it.
- Optional: Signed-off, Reported-By, Acked-By... just to give recognition.
Best practices

- Git pull, git rebase, etc... before committing.
  - Can be corrected later, but if you plan to push to a remote server, be synchronized with it.
- Don't edit the history (commits) unless it will be really necessary.
- Use branches as often as you can.
  - Branch for the stable version.
  - Branch for develop new features, or per new feature.
  - Branch to fix complex bugs, etc.
Best practices

- Sync with the remote server when you finish the corresponding feature, bug fix, development.
  - Your local repo is a good place but remember to share your finished work.
- If the remote server is controlled by an external person, consider to send him patches like the Linux kernel does.
  - Send one email per patch. Git-format-patch and git-send-email commands are your friends.
  - Tell him/her that he/she can pull from your repo.
More help

• Books:
  • “Pro Git” written by Scott Chacon. Available online at http://progit.org/book/

• Online tutorials: already given some links.

• Official website.

• Ask your nearest GIT expert :-P