

# Git: introduction

# Version control

- A system that is able to keep track of changes happening to files/directory, in order to possibly get a specific version of the file over time.
- Wanted characteristics of a VCS:
  - **Data integrity**: each small change shall be tracked, and never be lost
  - Possibility to get a **specific version** a file
  - **Speed**
  - **Collaboration**: the same files may be edited by several people

# Motivation

- <https://en.wikipedia.org/wiki/Git>
- Git is a version-control system, primarily used for source-code management in software development
- Git was created by Linus Torvalds in 2005 for development of the Linux kernel.
- Every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities
- Most of Git is written in C
- Compatible with Windows, macOS, UNIX

# You can create a Git repository without any remote counterpart

- Create a folder
- Do “**git init**”
- From that moment, the specific folder is a repository
- You can enter the `.git` subdirectory that contains specific configurations files.

# git status

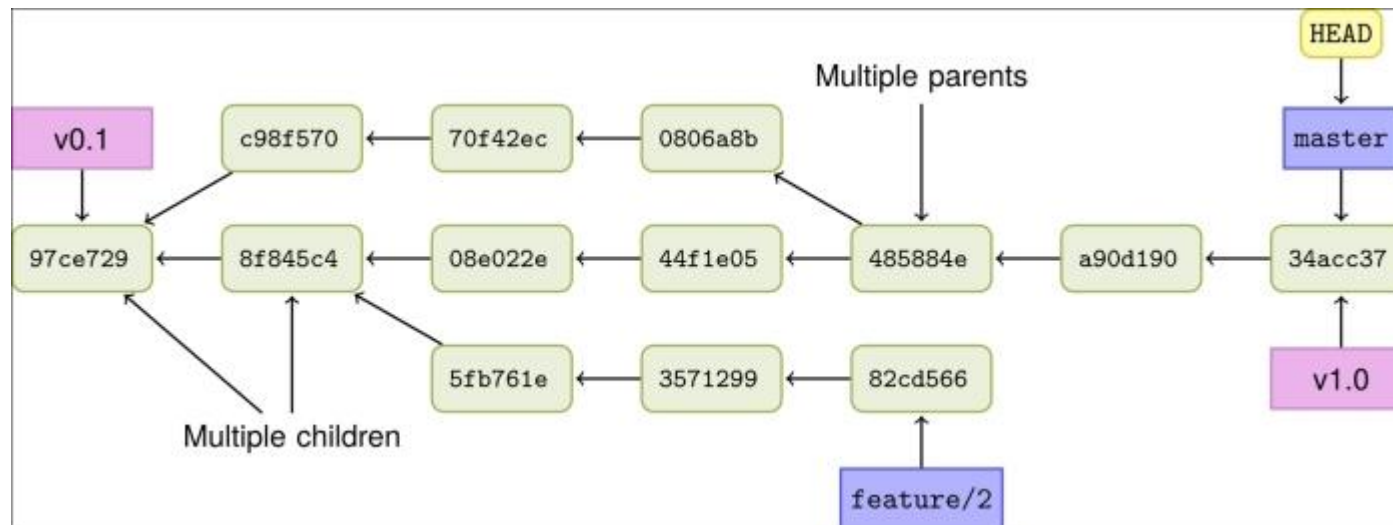
- Returns the status of the repository along with the list of files added/modified/removed/moved locally in comparison to the index.
- Very useful in identifying conflicting files
- A file / a list of files that were added/modified/removed/moved locally has to be manually added to the index through the basic commands “**git add**”, “**git rm**”, “**git mv**”

# Basic commands

- **git add** -> adds a file / a list of files to the index
- **git rm** -> removes a file / a list of files both locally and in the index
- **git mv** -> moves a file into a new location both locally and in the index
  
- **git reset** -> when some files have changed in the index, reset the situation (e.g. if I have added local file changes to the index, then resetting means make the index go back to the last committed version of the file)

# Git commit

- A commit operation saves the current index (possibly associating a name).
- Commits get saved inside the internal Git structure linked to a previous commit (it is a Directed Acyclic Graph).



“A branch is simply a lightweight movable pointer to one of these commits.”

A TAG is a fixed pointer/name given to a particularly important commit

# Git commit

- Correct way of committing is always providing a message describing the commit.
- **git commit -m "Description of the commit"**

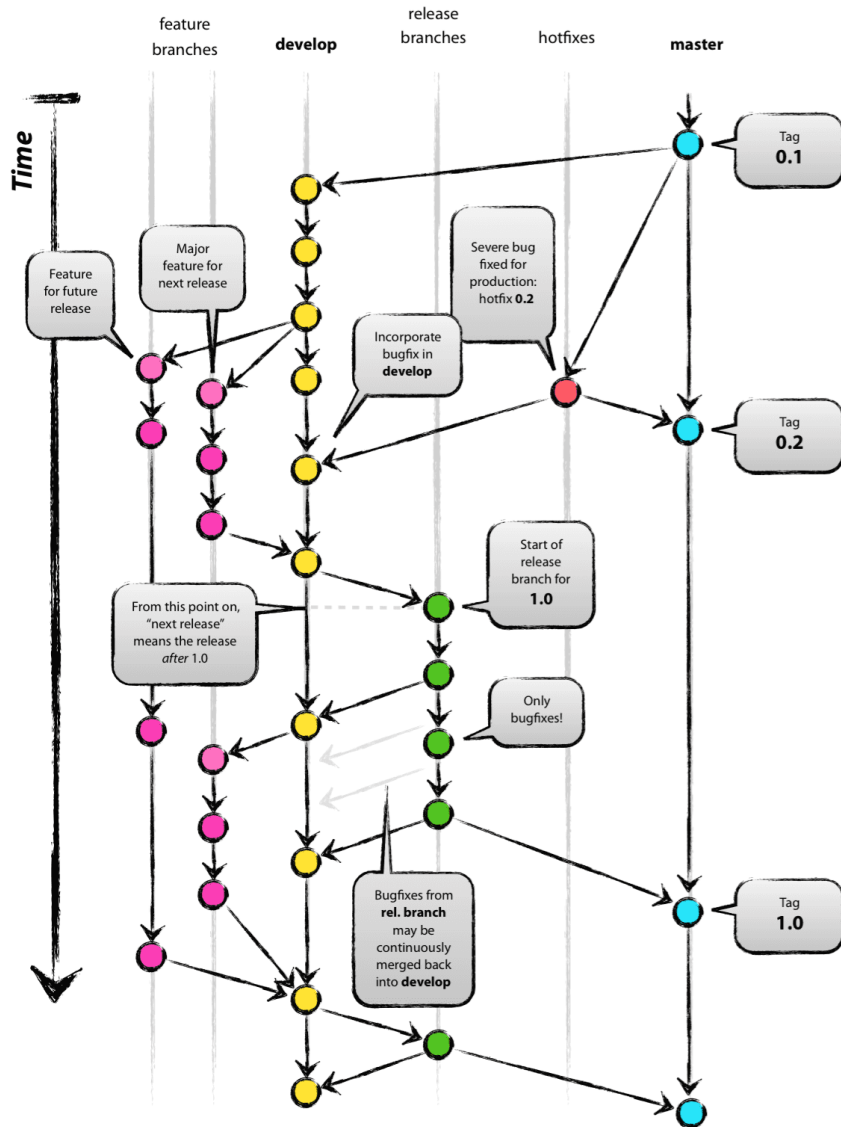


# Git branch

- “A branch is simply a lightweight movable pointer to one of these commits.”
- To see the list of branches that are saved into the local repository, then the:
- **git branch**
- command is of help.
- To move along branches that are existing on the system, it is enough to do **git checkout branch\_name**
- To create a new branch pointing to the commit that is “on head”, then it is enough to do **git checkout -b new\_branch\_name**

```
alpha-certification
archaic
bpmnIntegration
develop
hotfixes
master
release
* stochasticIntegration
```

# Why branches?



- To manage complex software projects where, for instance, multiple versions need to be maintained at the same time.
- Working on multiple “local” and “remote” branches could be useful to work in concurrency on different features (each feature may break something, so better to keep changes splitted).
- Branches may introduce new features, or contribute to solve a bug ...

# Committing on branches

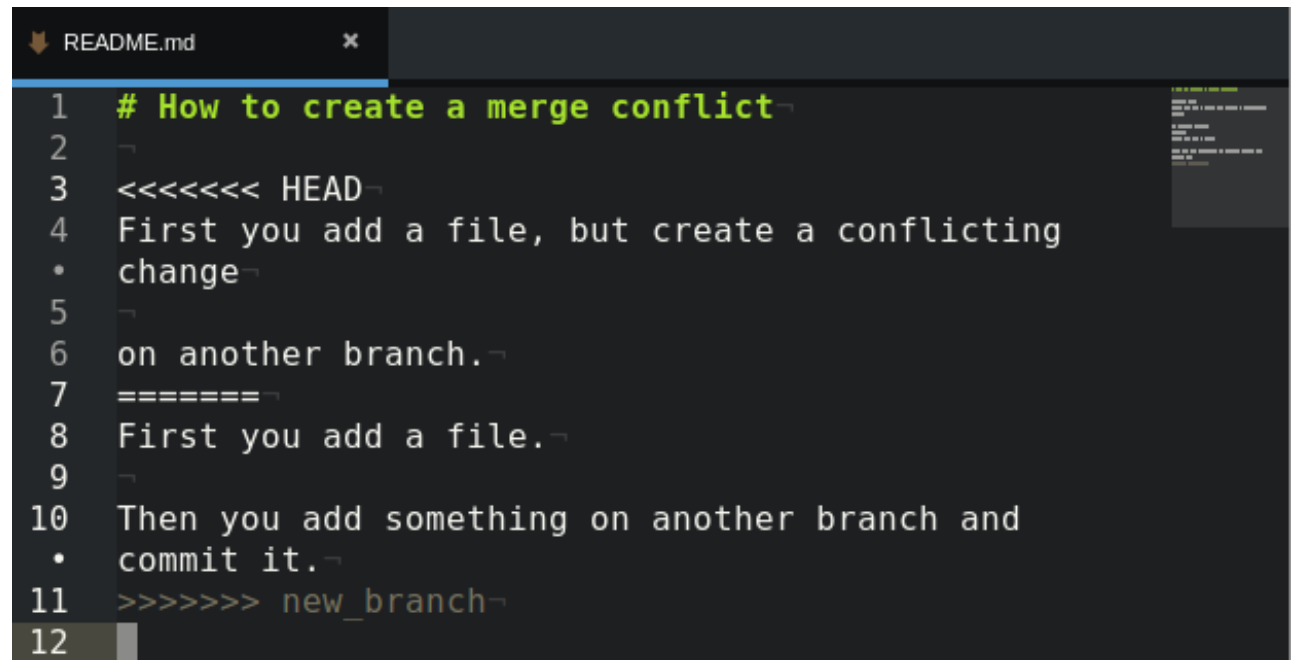
- A commit is added to the system pointing to the actual branch.
- The branch referral is then moved to the current commit.
- Merging permits to unite the contribute of different branches.
- Suppose we have a develop branch and different “feature” branches starting in different points of time.
- A merge operation from the feature branch to the develop branch shall take the contribution the feature branch and “unite” it with the develop branch (that could be changed from the moment the “develop” branch is created)

# Merge of branches

- Two things could happen:
  - The develop branch has not changed from the commit that started the “feature” branch .. In that case the develop branch is just moved to the current commit referred by the “feature” branch.
  - In the case the “develop” branch changes .. Then problems arises 😊
  - A new commit that “takes as input” both the last commit of the feature branch both the last commit of the develop branch has to happen.
  - When it happens automatically:
    - When touched files are different.
    - When the auto-conflict solver succeeds.
  - When it does not happen automatically:
    - When there are conflicts that the auto-merge strategy fails to resolve.

# How conflicts are signaled

- In **git status**, conflicting files are clearly signaled with the “Both modified” flag.
- To resolve a conflict, you have to open manually the file and fix all the discrepancies
- Then you can manually **commit**



```
README.md
1 # How to create a merge conflict
2
3 <<<<<<< HEAD
4 First you add a file, but create a conflicting
5 • change
6
7 on another branch.
8 =====
9 First you add a file.
10
11 Then you add something on another branch and
12 • commit it.
13 >>>>>>> new_branch
```

# How to do the merge of branches

- Go to the target branch through the command **git checkout target\_branch**
- Start the merge through the command **git merge branch\_to\_merge**
- If no conflict occur, then do nothing.
- If conflicts occur, then fix them and commit them manually.
- It is possible to delete from the list of branches a branch that has been already merged through the command **git branch -d branch\_name**.
- The command will protest if there are “unmerged changes” but will go on if everything has been merged correctly.

# Git log

- Reports the information about the commits:

Each commit has an unique identifier and reports clearly the branches that are pointing to it and the author

```
commit 33220552fc598a09e88ea95855191deb9c8fb73e (HEAD -> stochasticIntegration, origin/stochasticIntegration)
Author: Alessandro Berti <a.berti@pads.rwth-aachen.de>
Date: Tue Nov 20 16:45:51 2018 +0100

    Added Stochastic Petri Net example starting from Pandas dataframe

commit c975af44d90d551d7e1b7170861184923345f58b
Merge: b2f402a 0d56ad1
Author: Alessandro Berti <a.berti@pads.rwth-aachen.de>
Date: Tue Nov 20 16:41:07 2018 +0100

    Merge branch 'develop' into stochasticIntegration

commit 0d56ad1d821de0ca4bb4aea6fb4141f15e744131 (origin/release, origin/master, origin/hotfixes, origin/develop, origin/alpha-certification, origin/HEAD, release, master, hotfixes, develop, alpha-certification)
Author: Alessandro Berti <a.berti@pads.rwth-aachen.de>
Date: Tue Nov 20 16:39:53 2018 +0100

    Bug fix

commit b2f402a6e723a0722dd5fec29e636363c06363eb
Author: Alessandro Berti <a.berti@pads.rwth-aachen.de>
Date: Tue Nov 20 16:39:13 2018 +0100

    Added mechanism in order to get stochastic map also from dataframe/DFG elaborations

commit 78c9b18ce037ffe228b1f54da7f64a3c81ab7610
Merge: df59e29 ce2eda2
Author: Alessandro Berti <a.berti@pads.rwth-aachen.de>
Date: Tue Nov 20 16:27:43 2018 +0100

    Merge branch 'develop' into stochasticIntegration

commit ce2eda2d5b760c055e5ff827284a984bb7a77e97
Author: Alessandro Berti <a.berti@pads.rwth-aachen.de>
Date: Tue Nov 20 16:26:05 2018 +0100

    Bug fix

commit df59e298151abffdd17cf7508247d18f2952655e
Merge: 05ab813 468e344
Author: Alessandro Berti <a.berti@pads.rwth-aachen.de>
Date: Tue Nov 20 16:11:16 2018 +0100

    Merge branch 'develop' into stochasticIntegration

commit 468e34463b965f6b1866c151673ea3f647db3073
Author: Alessandro Berti <a.berti@pads.rwth-aachen.de>
Date: Tue Nov 20 16:08:33 2018 +0100
```

# Log for a particular file

- Using the command **git log --follow nomefile**

```
PS C:\Users\berti\pm4py-source> git log --follow .\setup.py
commit 1049cab9c3105e0783d9892d3291f593be8f5203
Author: Alessandro Berti <a.berti@pads.rwth-aachen.de>
Date:   Fri Nov 16 14:56:30 2018 +0100

    Revised set-up with process tree packages

commit d8e00fee89921c435dd86eee716d1910e7d11242
Author: S.J. van Zelst <s-j-v-zelst@users.noreply.github.com>
Date:   Fri Nov 2 16:52:40 2018 +0100

    Update the email of the author

    Let's not give me all the credits for Ale's work :-)
```

```
commit 5a0c99e5f9f8412b416784a3d4ace0aa7971d8d0
Author: Alessandro Berti <javert@northwood.northwood.net>
Date:   Fri Nov 2 08:35:34 2018 +0100

    Removed transition system visualizer from release. Added setup.py file.
```



# How to “delete” a local commit

- Sometimes we commit things that “we aim to have never committed”.
- While physically the commit remains, it is possible to leave it as unconnected nodes so future commits do not start from it.
- To “delete” in this way a local commit (that has not been pushed to remote) it is enough to do **git reset --hard sha1ofcommit**
- Then, changes have to be committed. This commit would take the lead for the current branch.

# How to retrieve a particular version of a file

- We have seen that the command **git checkout** accepts the name of a branch. But a branch is effectively corresponding to a particular commit so it's possible to do **git checkout shaidcommit** in order to retrieve the status of the repository at the given commit.
- It's also possible to retrieve a specific version of a file with the command **git checkout shaidcommit interestingfile**
- Changes to that version of **interestingfile** could be then committed to the current branch.



# How to know who to blame inside your team for a bug 😊

- It is enough to do **git blame file\_name**
- Each row has an assignee so it is clear who have added the particular row

# Important file: .gitignore

- For certain files, it may be useless to include them in the version control:
  - Saved output files (may be big)
  - Compiled files (.class, .o)
- A **.gitignore** file could contain the names of the files not to include in the **git status** representation, or some regular expressions (e.g. **\*.o** for avoiding **.o** files to be included)

# Submodules

- <https://git-scm.com/docs/git-submodule>
- Include physically other projects inside your project by reference.
- The simplest deployment could be **git submodule add repository\_url**
- Then, a **.gitmodules** file is created and the **config** file inside the **.git** directory is updated with information about the position of the repository.
- Following that, these instructions need to be deployed:
  - **git submodule init**
  - **git submodule update**

# Submodules

- If the following commands are deployed:
- **git submodule add** <https://github.com/pm4py/pm4py-source.git>
- **git submodule init**
- **git submodule update**
- Then a **pm4py-source** folder is created where, in the internal content, the PM4Py project is included.
- If we operate inside the pm4py-source, commits are touching THAT repository.

# Remote branches

- For now, we have worked on local branches.
- Even at local level, git is powerful enough to ease developers life:
  - Version control
  - Branches
- But the power of **git** is to enable collaboration among people of a team.
- A remote repository is a repository that is not necessarily on your local machine, that can sync the local repository or be synced from the local repository.
- Ideally, remote repositories should be easily accessible by the members of the team (notable examples could be Github, Bitbucket, Gitlab ...)



# How to “get sync” with a remote branch

- **git clone** operation clones an exact copy of the remote repository in the local machine. All the history of commits is included in the local repository.
- It is possible to “connect” a local repository with a remote repository in a later stage. In that case, the following command needs to be deployed:
- **git remote add origin urloftherepository**
- After adding a remote reference to the repository, it is possible to pick its commits (and so, the branches) with the command **git fetch -all**.
- These would not be added automatically to the local branches. If that is intended, then the command **git pull -all** is what is needed.

# Git fetch --all

- You can see both local and remote branches.

```
alpha-certification
archaic
bpmnIntegration
* develop
hotfixes
master
release
stochasticIntegration
remotes/origin/HEAD -> origin/master
remotes/origin/alpha-certification
remotes/origin/archaic
remotes/origin/bpmnIntegration
remotes/origin/develop
remotes/origin/hotfixes
remotes/origin/master
remotes/origin/release
remotes/origin/stochasticIntegration
```

# PULL vs PUSH

- **PULL operation:** I take remote commits to my local repository
- **PUSH operation:** I take local commits and I send them to the remote repository.
- It's always better to pull before pushing 😊
- If there are commits in the remote repository that are not in the local repository, the push operation will fail and we are reminded to pull
- When the pull happens, if there are new commits in the remote repository, the “commit DAG” is changed accordingly.
- As the merge of branches, also the pull can cause conflicts.
- Conflicts need to be resolved in the same way.

# Pushing from a local branch that does not exist remotely

- In this case, **git push** itself would fail.
- The right command is: **git push --set-upstream origin namebranch**
- This creates a correspondence between the **namebranch** (local branch) and the remote branch **origin/namebranch**

# Remove a commit that has been pushed to the remote repository

- In this case, the command **git revert shaofthecommit** shall be used.
- A commit shall happen, where files of the commit are reverted to the previous status.
- The old commit would still appear in the **git log** but will then look unconnected in the DAG (so in practice they won't turb lives anymore).

# Removing a remote branch

- After merging (through **git merge**) and removing the local branch (**git branch -d ...**) one may wonder how to remove the branch from the remote repository.
- Actually, this is possible through the command **git push origin :deletedbranchname**
- To delete locally a branch that has been removed remotely, it is possible to use the command **git fetch -p**

# Example: Github

- On Github, I can create a new remote repository. After that, I could clone it and work as described.
- If I need to work on a project to which I do not have access privileges, I could:
  - Fork it
  - Do the changes on “my own” repository (that is a clone of the other repository).
  - Do a pull request
- A pull request is a request that reaches the creator(s) of the original project in order to make your changes accepted.