



Introduction to Git

[What is version control?](#)

[Life cycle of files](#)

[Ignore files](#)

[Java.gitignore](#)

[JENKINS.gitignore](#)

[C++.gitignore](#)

[What is a Branch?](#)

[Git on CLI](#)

[GitFlow](#)

[Main branches: Master and Main](#)

[Main branches: Develop](#)

[Auxiliary branches: Feature](#)

[Auxiliary branches: Release](#)

[Auxiliary branches: Hotfix](#)

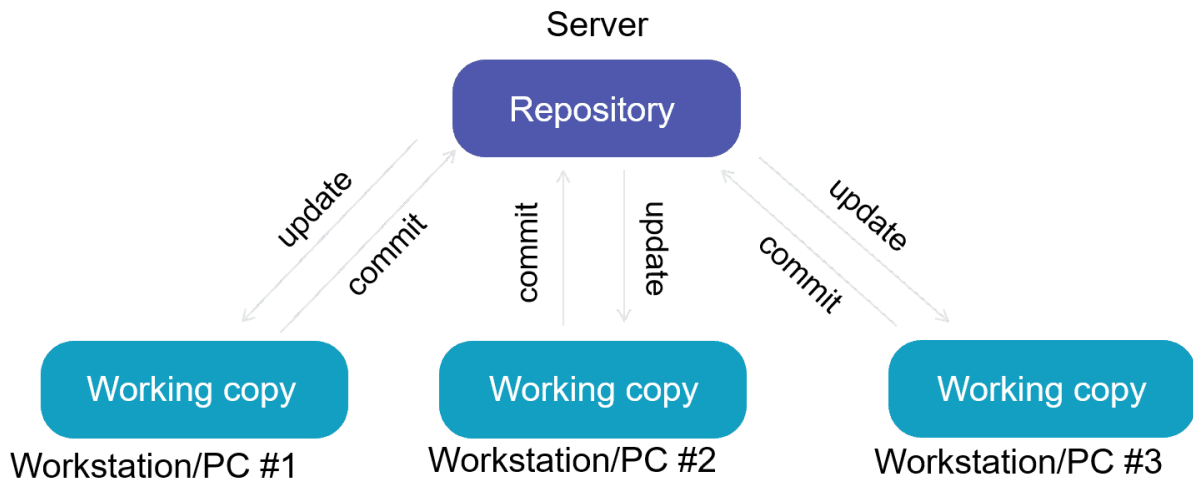
[Basic commands in GIT](#)

[**Knowledge test**](#)

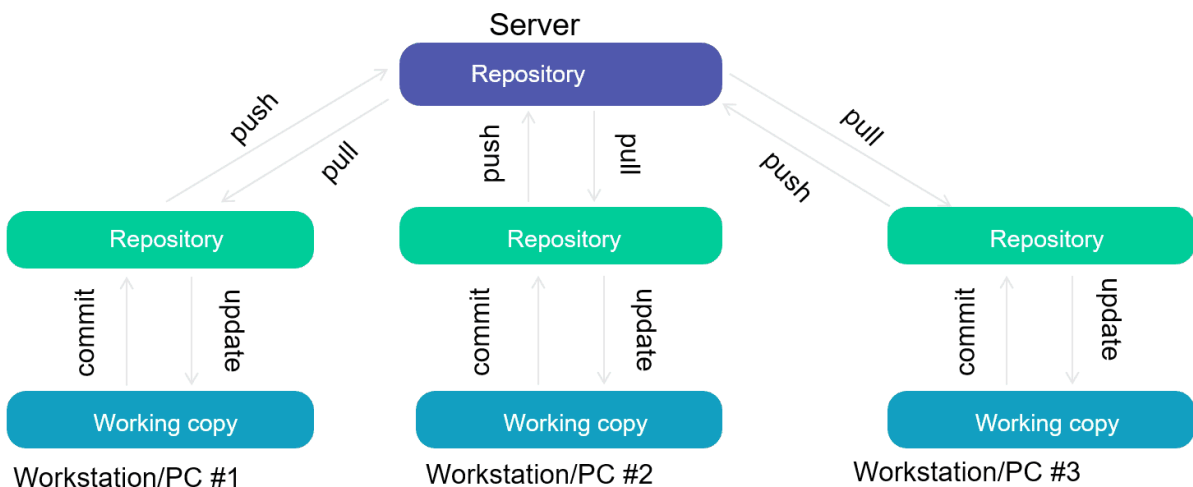
What is version control?

- Version control is the **management of the changes that are made to the components of a product or a product configuration**. A version, revision or release of a product is the status in which the product is at a given point in its development or modification.
 - Although a version control system can be conducted manually, it is highly advisable to have tools that facilitate management, giving rise to the **VCS** (*Version Control System*).
 - Using version control software **is a best practice** for high-performing software and **DevOps** teams because it helps developers move faster and enables software teams to maintain **efficiency** and **agility** as the team **scales** to include more developers.
-

Centralized Version Control System



Distributed Version Control System

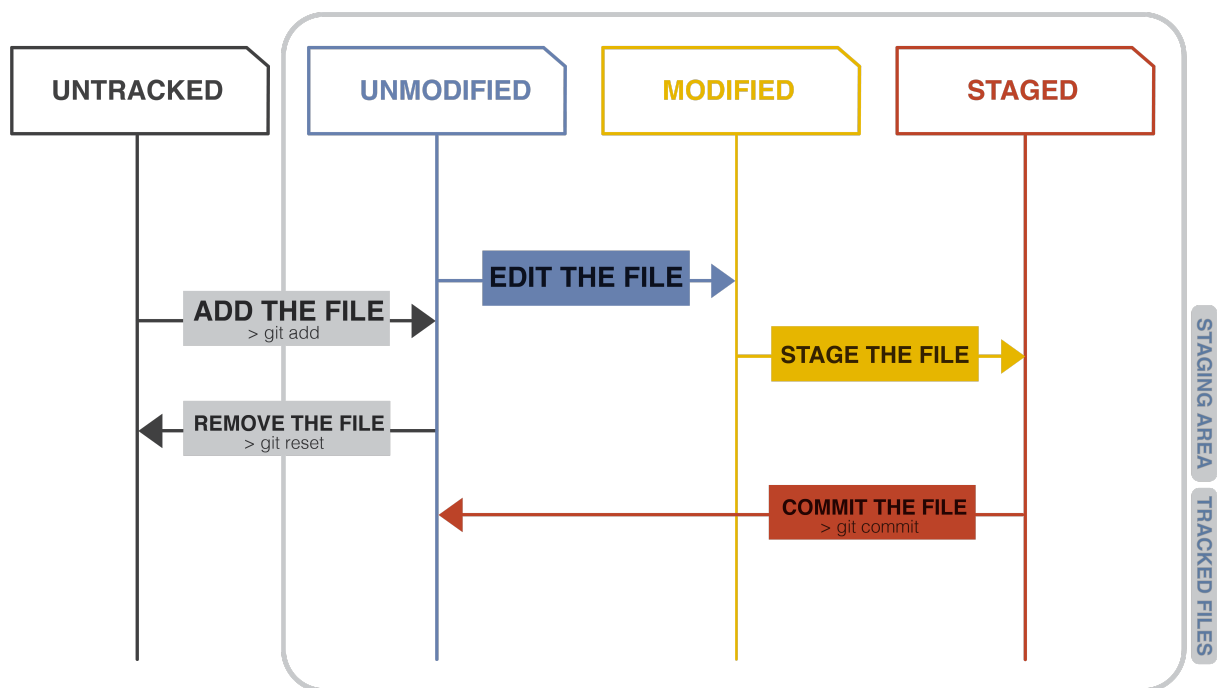


- Each user has a local repository.
- Different repositories can exchange and mix revisions amongst themselves.
- It is common to use a repository that is normally available, which serves as a synchronisation point for the different local repositories.
- Examples: **Git** and Mercurial.

Life cycle of files

In Git, files can be found in the following statuses:

- **Untracked** files: are files that do NOT live inside Git, only on the hard disk. They have never been affected by Git *add*, so Git has no record of their existence. That is, they have not been added to any repository.
- **Tracked** files: the file has been added to a repository.
- **Staged** files: the file has been added to the repository, but has not yet been committed to it.
- **Modified** files: the file has been modified.
- **Unmodified** files: the file has not been modified.
- **Committed** files: the file has already been updated in the repository.



Ignore files

.gitignore

- Create a text file and name it ".gitignore" (remember to put the "." at the beginning).
- Then edit this file as necessary.
- Each new line must include an additional file or folder that you want Git to ignore.

Patterns of matching entries

Entries in this file may also follow a matching pattern:

'*' is used as a match or wildcard.

/ is used to ignore paths relative to the .gitignore file.

is used for adding comments to the file.

```
#Ignore Mac system files
.DS_store

#Ignore the folder node_modules
node_modules

#Ignore all text files
*.txt

#Ignore files related to API keys
.env

#Ignore SAAS configuration files
.sass-cache
```

Generic .gitignore file format

Java.gitignore

24 lines (19 sloc) | 290 Bytes

```
1  # Compiled class file
2  *.class
3
4  # Log file
5  *.log
6
7  # BlueJ files
8  *.ctxt
9
10 # Mobile Tools for Java (J2ME)
11 .mtj.tmp/
12
13 # Package Files #
14 *.jar
15 *.war
16 *.nar
17 *.ear
18 *.zip
19 *.tar.gz
20 *.rar
21
22 # virtual machine crash logs, see http://www.java.com/en/download/help/error_hotspot.xml
23 hs_err_pid*
24 replay_pid*
```

JENKINS.gitignore

50 lines (40 sloc) | 1.77 KB

```
1  # Learn more about Jenkins and JENKINS_HOME directory for which this file is
2  # intended.
3  #
4  # http://jenkins-ci.org/
5  # https://wiki.jenkins-ci.org/display/JENKINS/Administering+Jenkins
6  #
7  # Note: secret.key is purposefully not tracked by git. This should be backed up
8  # separately because configs may contain secrets which were encrypted using the
9  # secret.key. To back up secrets use 'tar -czf /tmp/secrets.tgz secret*' and
10 # save the file separate from your repository. If you want secrets backed up
11 # with configuration, then see the bottom of this file for an example.
12
13 # Ignore all JENKINS_HOME except jobs directory, root xml config, and
14 # .gitignore file.
15 /*
16 !/jobs
17 !/.gitignore
18 !/*.xml
19
20 # Ignore all files in jobs subdirectories except for folders.
21 # Note: git doesn't track folders, only file content.
22 jobs/**
23 !jobs/**/
24
25 # Uncomment the following line to save next build numbers with config.
26
27 #!jobs/**/nextBuildNumber
```

C++ .gitignore

32 lines (26 sloc) | 270 Bytes

```
1  # Prerequisites
2  *.d
3
4  # Compiled Object files
5  *.slo
6  *.lo
7  *.o
8  *.obj
9
10 # Precompiled Headers
11 *.gch
12 *.pch
13
14 # Compiled Dynamic libraries
15 *.so
16 *.dylib
17 *.dll
18
19 # Fortran module files
20 *.mod
21 *.smod
22
23 # Compiled Static libraries
24 *.lai
25 *.la
26 *.a
27 *.lib
```

When I make a commit, can my changes be seen by other users?

- No

Other users cannot see your modifications because the **commits** go to the local repository.

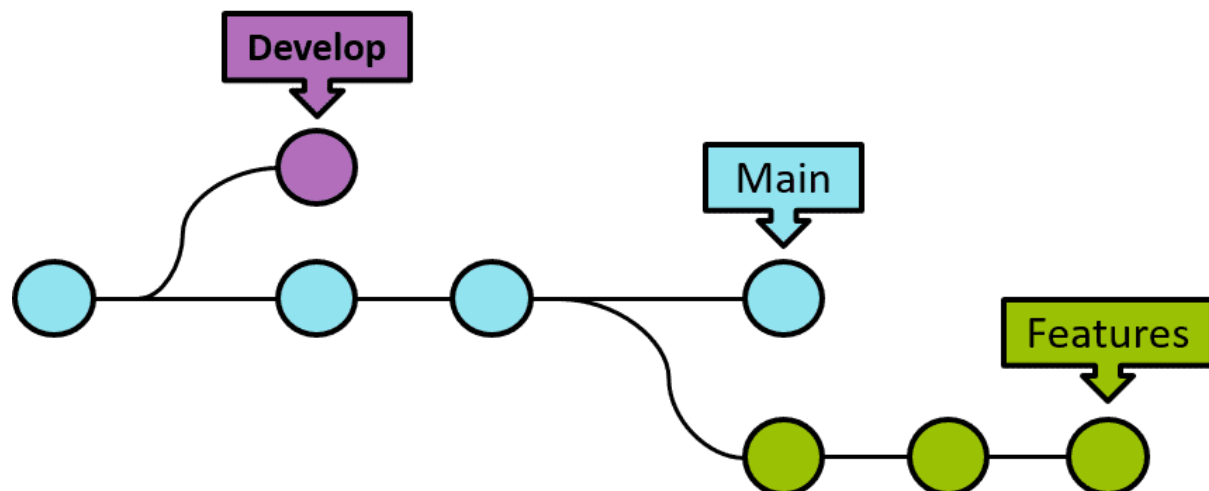
For other users to see your changes, you have to upload them to the remote repository using the **push** command.

Can I make a commit offline?

- Yes

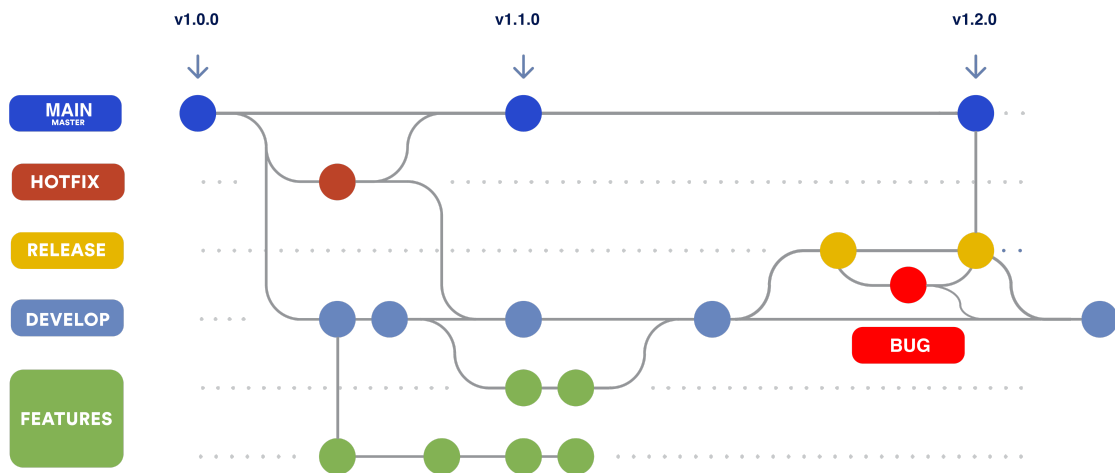
Offline I can commit the changes we have in our staging area, make a historical comparison of the file I want to modify, check the file history, access any code revisions, merge branches, etc.






What is a Branch?



- A **branch** is a version of our code; it is a fork or division within your project where you can make changes without modifying the main branch.
- Creating branches allows us to work on different versions of the same file, test changes without affecting the project and then, once we have checked that these changes work correctly, we can incorporate them into the main branch if we wish.
- **In short, a branch is nothing more than a name given to a commit, from which you start working independently and to which new commits from the same branch will be linked. Branches can be mixed so that all work done in one branch becomes part of another.**
- As a general rule, a **main** is considered the main branch and the root of most other branches.
- Most commonly, the "final code" is found in **main**, which then goes to production. It is the branch where all the other branches are mixed sooner or later to finalise a task and incorporate it into the end product.
- The diagram shows a repository with two isolated development pipelines, one for a small function (**Develop**) and one for a larger function (**Features**).

- By developing them in branches, we can not only work with both in parallel, but also avoid dubious code from being merged into the main branch (**Main**).



Graph	Commit
	bdf09f20   master  origin  origin/HEAD Fix object ext b415f6aa Merge pull request #335 from PatrykPorabik/master cd0efe6d Added GrandNode 83405628 Standard filters (#315) b1ce9761 Added other Ruby's date formats (#314) 6068a1c8 Support to %s dateformat (#313) b83c2cb9 Sorting when only some elements contain the sorting c 24491a68 Merge pull request #280 from watkinsmatthewp/278_v 93645dd7 Merge pull request #297 from nytmo/patch-2 fea10c4b Merge pull request #301 from Tewr/patch-1 7599a748 Merge pull request #302 from vitalragaz/DictionaryIter 9d1261ea Update to support decimal in "times" filter (#306) ba477835 Fix for TestVariableNotFoundFromAnonymousObject t 3ffb083d Changed Dictionary Handling, So that an iterator is av d8d63758 Update README.markdown 6bfb60b8 Slice filter tests 6be1a0b0 Slice filter resilient to negative start index out of bound 0d7cb51b Sort filter made resilient to null input (#294) baaf2a6f Date filter - "now" reserved string should return correc

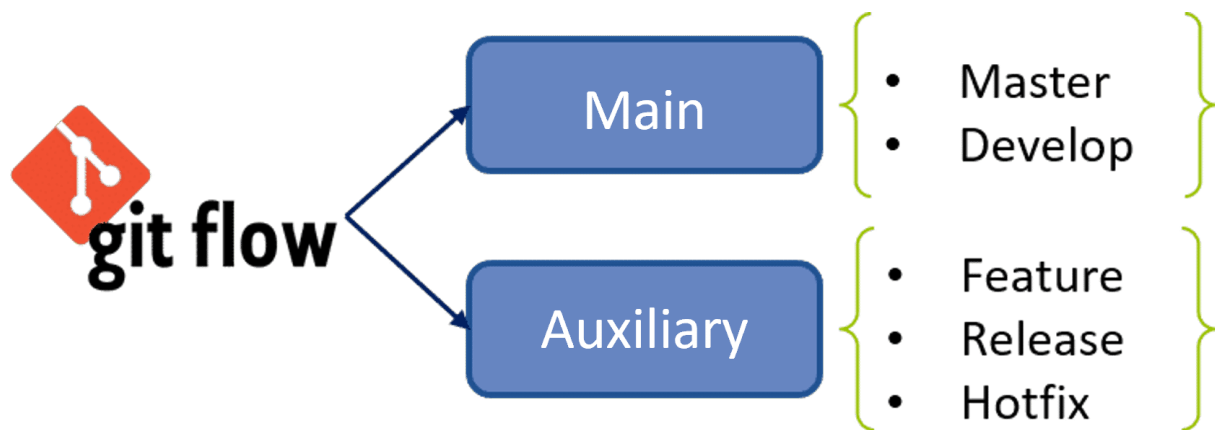
- Git keeps a **pointer** to the version we're in, so we know which version we're working on. **This pointer is called HEAD.**

Git on CLI

- **git branch <branch>:** Creates a new branch with the name <branch> in the repository from the last commit, i.e. where the HEAD points to.
- **git branch:** shows the active branches of a repository indicating with * the currently active branch.
- **git log:** shows the operations that have been performed.
- **git log --graph:** shows graphically how branches fork and rejoin. Including all branches (-all).
- **git checkout <branch>:** Updates the files in the working directory to the latest version of the repository corresponding to the branch <branch>, and activates it, i.e. the **HEAD** points to this branch's latest **commit**.

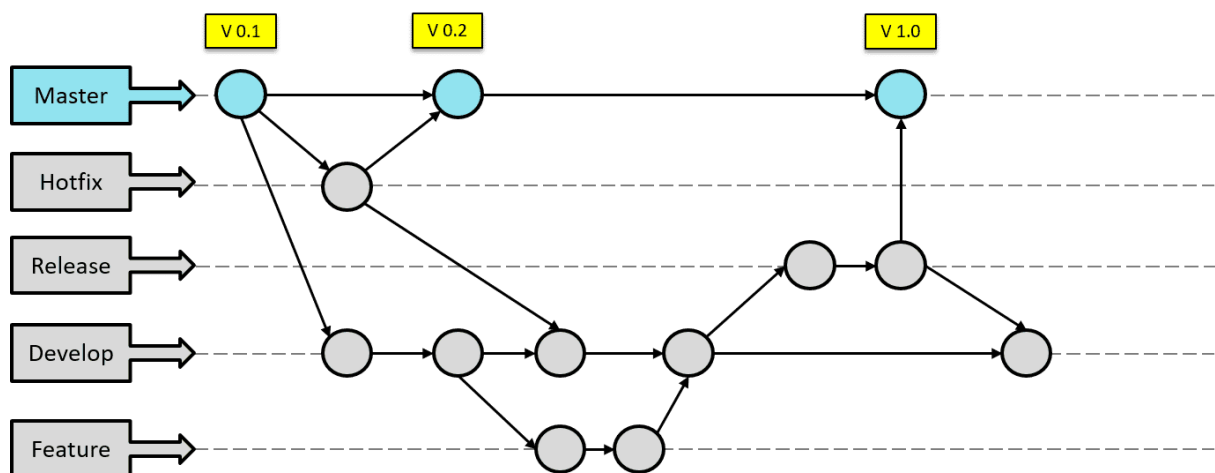
- **git merge <branch>**: Integrate changes from the branch <branch> into the current branch pointed to by the **HEAD**.
- **git branch -d <branch>**: removes the branch named <branch> as long as it has been merged previously.
- **git branch -D <branch>**: deletes the branch with name <branch> even if it has not been merged. If the branch has not been merged previously, all changes to that branch will be lost.

GitFlow



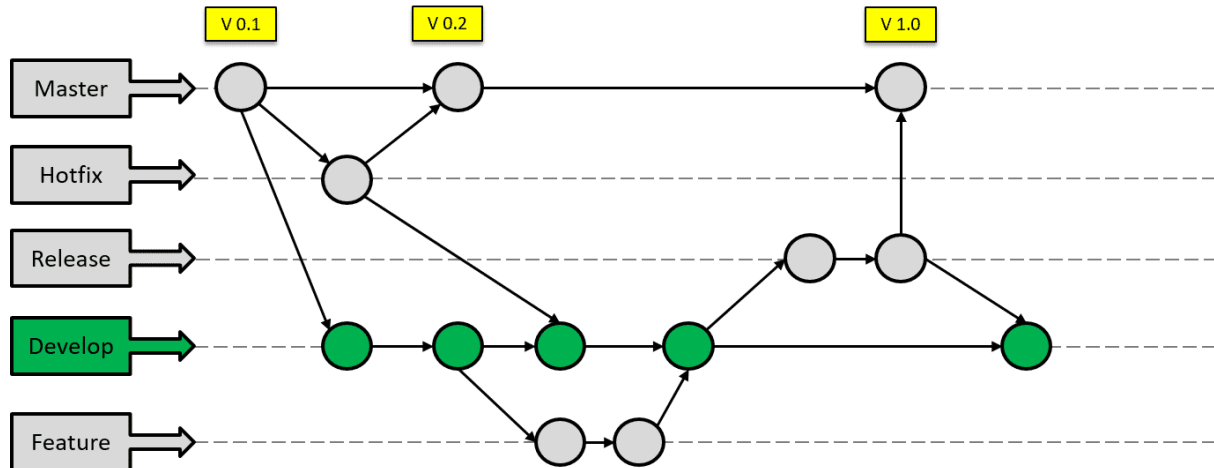
Main branches: Master and Main

- This always contains code in a state ready to be deployed in production (**master**). No errors.



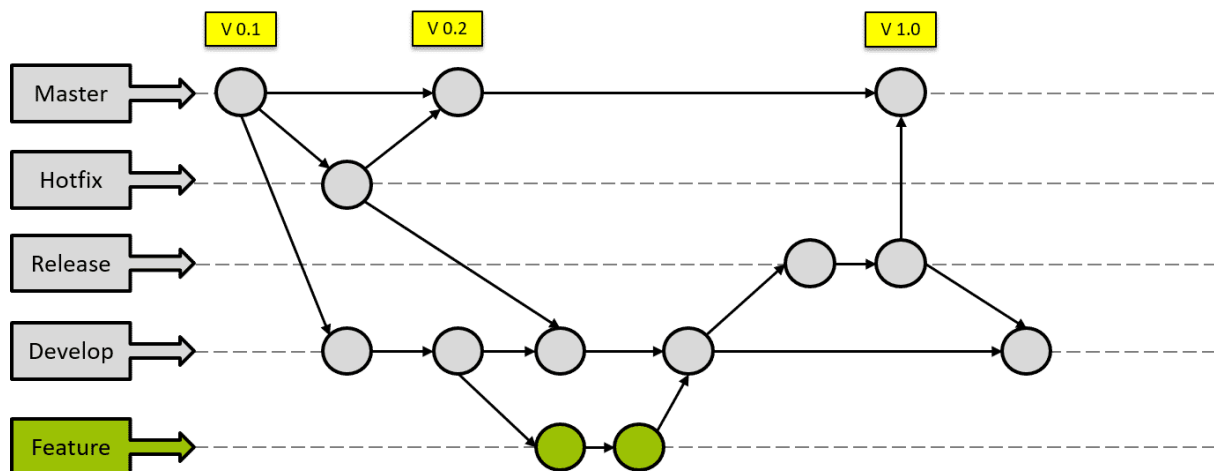
Main branches: Develop

- It contains the latest changes developed for the next version of the software.
- It does not necessarily contain full functionality, but it must be compilable (**develop**).



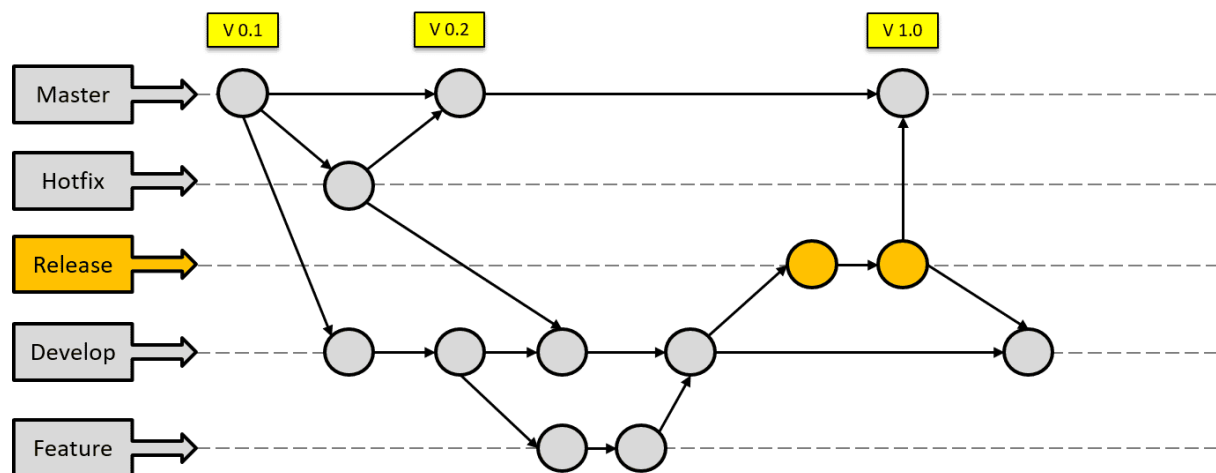
Auxiliary branches: Feature

- These will contain a new development or upgrade; each new improvement or feature that we introduce in our software will have a branch that will contain its development.
- They stem from the **develop** branch and once the development of the enhancement is completed, they are integrated back into the **develop**.



Auxiliary branches: Release

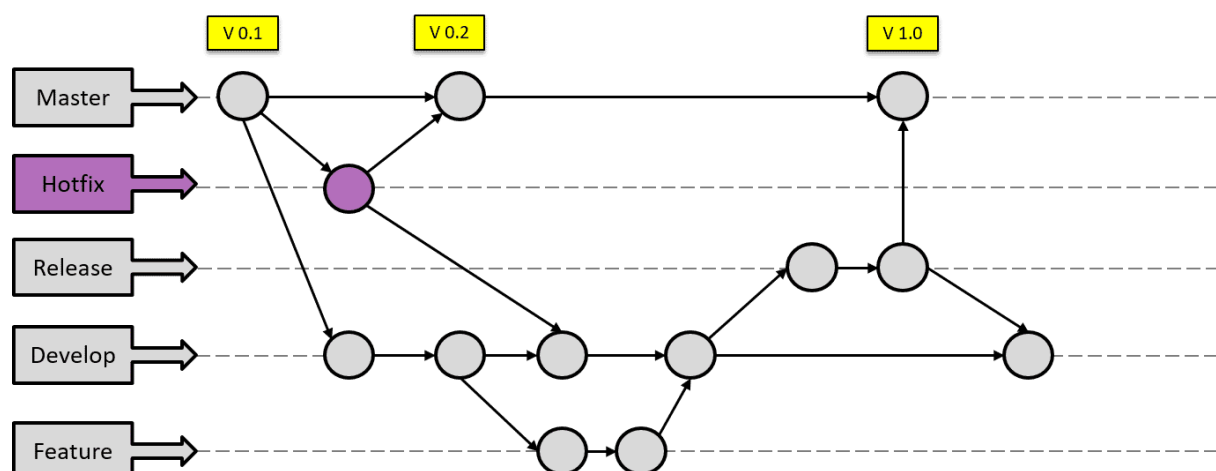
- They are created when the next version of the software is going to be released and they stem from the **develop** branch.
- As a result, development of new features is frozen, and work is done on bug fixes and documentation.
- Once it is ready for publication, it is integrated in **master** and labelled with the corresponding version number.



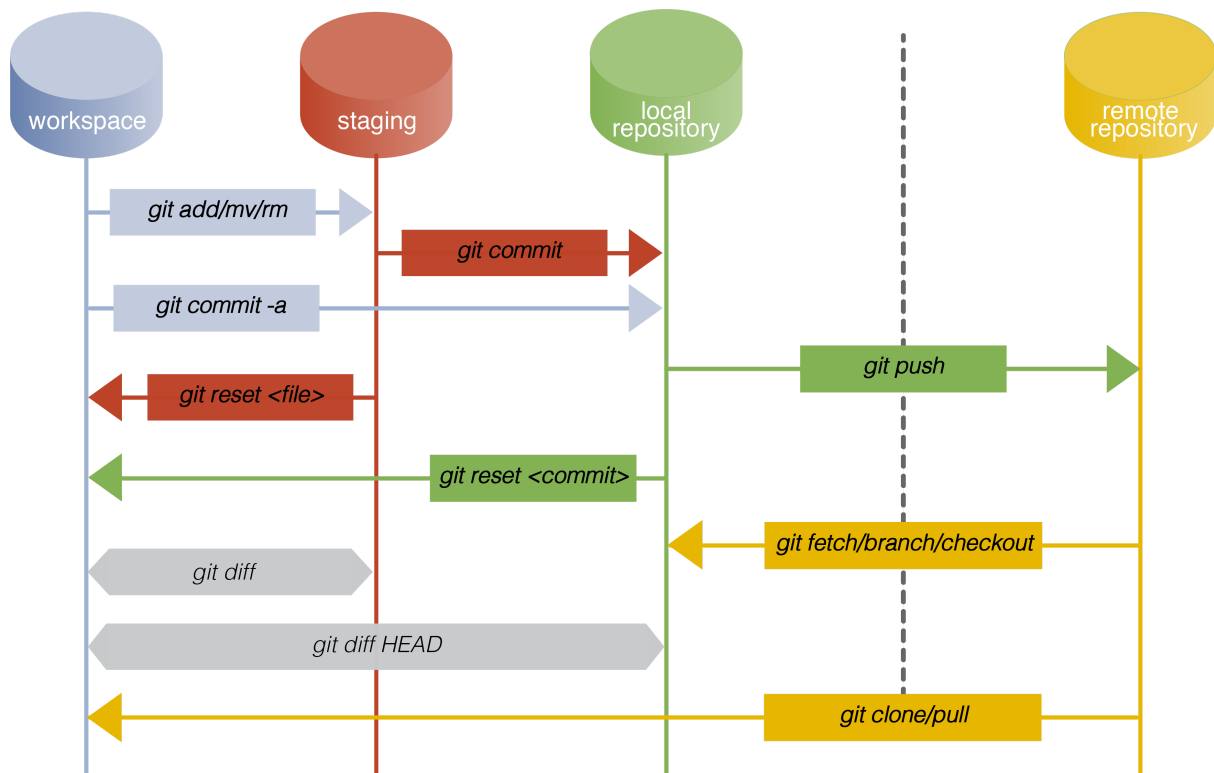
Auxiliary branches: Hotfix

If our code contains critical bugs that need to be patched immediately, we can create a **hotfix** branch from the corresponding release in **master**.

This branch will contain only the changes that need to be made to patch the bug and fix it, they are not planned and should be fixed as soon as possible.



Basic commands in GIT



Knowledge test

After initialising a GIT repository and creating a file named 'test.txt'. Which of the following commands will not work?

- `git commit -m "first file"`

Which command will save the changes from my 'myfile.txt' file to my local Git repository?

- `git add myfile.txt + git commit -m "my recent changes"`

If you want to get a copy of an existing Git repository in GitLab, the command you need is:

- `git clone`

What is the default text editor for the GIT console with a Git installation on Windows?

- Vim

Git is a centralised version control software.

- False
- Git is a distributed version control system, not a centralized one.
- This means that every developer's working copy of the code is also a repository that can contain the full history of all changes, unlike centralized version control systems, where a single central repository is the source of all versions and history of the project.

After installing GIT and before making the first commit, which two configuration properties must be complete?

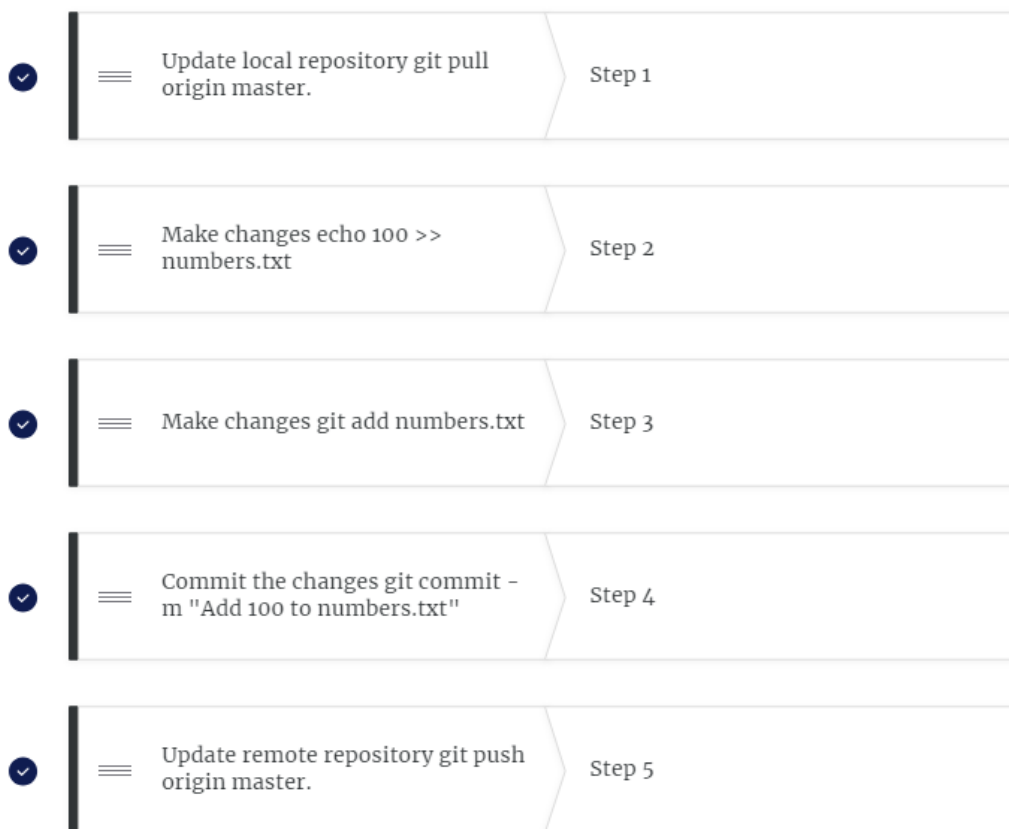
- Username and email address

The three main sections of a Git project are: Working directory, staging area and repository.

- False

In what order should you do these actions to minimise the possibility of conflict?

In what order should you do these actions to minimise the possibility of conflict?



Shortcut provided by git, if you want to skip the staging area.

- git commit -a

Information about what is going into your next revision (commit) is stored in _____.

- Staging Area

How does "git push" differ from "git commit"?

- When we push changes, we are interacting with a remote repository to update it with the changes we have made locally.
- In contrast, commit only updates your local repository.

Which command is used to list the changes (commits) you have made to the repository?

- `git log`

What is the shortest rule in `.gitignore` to ignore all **.data** files in `result/data/position/gps` given the following structure? ****Do not ignore the info.txt file***

`results/data/position/gps/a.data`

`results/data/position/gps/b.data`

`results/data/position/gps/c.data`

`results/data/position/gps/info.txt`

`results/plots`

- `results/data/position/gps/*.data`

Which of the following commands will create a new local branch called `Develop`?

- `git checkout -b develop`

Lists all locally available branches.

- `git branch`