Control de Versions

DAM 1 M05 UF2

21 d'abr. de 2023

Javier Garcia Vera

ÍNDEX

1 Introducció

En aquest document es tractaràn diferents apartats sobre la gestió de versions, els seus avantatges, desavantatges, dificultats, ajudes, i més informació sobre una eina que, ben utilitzada, agilitza molt la producció sobre tot en entorns de cooperació.

Abans de començar cal dir que hi ha apartats objectius i subjectius en aquest document, tota la informació dins del document està subjecte a canvis i pot quedar obsoleta ràpidament.

Ara si, comencem amb un apartat subjectiu, una experiència personal.

2 La meva Història de por

La meva història de por va ser quan estava fent un projecte tècnic al grau mitjà utilitzant Microsoft Sharepoint. Els majors problemes que vam tenir van ser que cadascú treballava quan podia des d'on podia, això va significar que alguns vam estar treballant al mateix document a la mateixa vegada i algú potser ho volia corregir aquella nit.
Ho vam gestionar fent servir Sharepoint i Onedrive, de manera que la versió del document que es carregava al obrir de manera local era la última que vam tenir quan hi havia conexió al Sharepoint.

Això ens va fer adoptar una manera de control de versions una mica rudimentària, quan modificàvem un arxiu, el guardàvem amb el nom de l'arxiu, el nom de qui ho ha modificat i el dia i la hora de la modificació, i després quan ens posàvem d'acord a clase revisavem els canvis i els implementàvem agafant-los des de els documents guardats amb el format explicat i posant-los on correspon al document principal o Main.

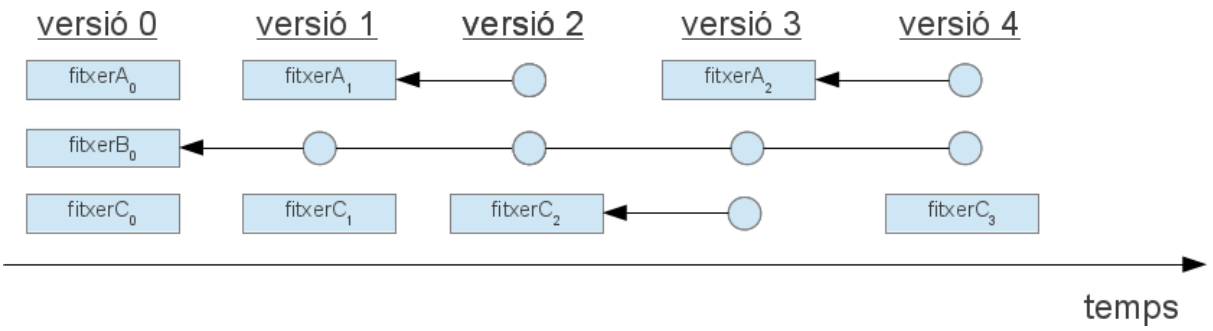
Això, encara que era una mica laboriós, significava que mai no es perdia ni canviava el document main sense saber-ho els membres del grup. També significaba que en cas que es volgués revertir un canvi a un dels documents no main sempre es podia copiar des de la última versió del document main.

Aquesta situació va durar aproximadament 2 mesos abans de que acabéssim el projecte. Es va entregar el document main sense cap problema

3 Comparació de SVCs

Sistemes de control de versions: CVS
Sistemes de control de versions: Git

4 Quantes versions guardem



versió	nombre de fitxers guardats
0	3
1	2
2	1
3	1
4	1

5 Configuració global

user.name=Javier Garcia Vera

user.email=cf22javier.garcia@iesjoandaustria.org

core.repositoryformatversion=0

core.filemode=true

core.bare=false

core.logallrefupdates=true

remote.origin.url=git@github.com:Javiergave/introM3

remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*

branch.main.remote=origin

branch.main.merge=refs/heads/main

pull.rebase=false

6 Comandes útils

NAME

git-clone - Clone a repository into a new directory

DESCRIPTION

Clones a repository into a newly created directory, creates remote-tracking branches for each branch in the cloned repository (visible using git branch --remotes), and creates and checks out an initial branch that is forked from the cloned repository's currently active branch.

After the clone, a plain git fetch without arguments will update all the remote-tracking branches, and a git pull without arguments will in addition merge the remote master branch into the current master branch, if any (this is untrue when "--single-branch" is given; see below). This default configuration is achieved by creating references to the remote branch heads under refs/remotes/origin and by initializing remote.origin.url and remote.origin.fetch configuration variables.

NAME
git-init - Create an empty Git repository or reinitialize an existing one

DESCRIPTION
This command creates an empty Git repository - basically a .git directory with subdirectories for objects, refs/heads, refs/tags, and template files. An initial branch without any commits will be created (see the --initial-branch option below for its name). If the $GIT_DIR environment variable is set then it specifies a path to use instead of ./.git for the base of the repository. If the object storage directory is specified via the $GIT_OBJECT_DIRECTORY environment variable then the sha1 directories are created underneath - otherwise the default $GIT_DIR/objects directory is used.
Running git init in an existing repository is safe. It will not overwrite things that are already there. The primary reason for rerunning git init is to pick up newly added templates (or to move the repository to another place if --separate-git-dir is given).

NAME
git-add - Add file contents to the index

DESCRIPTION

This command updates the index using the current content found in the working tree, to prepare the content staged for the next commit. It typically adds the current content of existing paths as a whole, but with some options it can also be used to add content with only part of the changes made to the working tree files applied, or remove paths that do not exist in the working tree anymore.

The "index" holds a snapshot of the content of the working tree, and it is this snapshot that is taken as the contents of the next commit. Thus after making any changes to the working tree, and before running the commit command, you must use the `add` command to add any new or modified files to the index.

This command can be performed multiple times before a commit. It only adds the content of the specified file(s) at the time the add command is run; if you want subsequent changes included in the next commit, then you must run `git add` again to add the new content to the index.

The `git status` command can be used to obtain a summary of which files have changes that are staged for the next commit.

The `git add` command will not add ignored files by default. If any ignored files were explicitly specified on the command line, `git add` will fail with a list of ignored files. Ignored files reached by directory recursion or filename globbing performed by Git (quote your globs before the shell) will be silently ignored. The *git add* command can be used to add ignored files with the `-f` (force) option.

NAME

git-mv - Move or rename a file, a directory, or a symlink

DESCRIPTION

Move or rename a file, directory or symlink.

git mv [-v] [-f] [-n] [-k] <source> <destination>

git mv [-v] [-f] [-n] [-k] <source> ... <destination directory>

In the first form, it renames <source>, which must exist and be either a file, symlink or directory, to <destination>. In the second form, the last argument has to be an existing directory; the given sources will be moved into this directory.

The index is updated after successful completion, but the change must still be committed.

NAME

git-reset - Reset current HEAD to the specified state

DESCRIPTION

In the first three forms, copy entries from `<tree-ish>` to the index. In the last form, set the current branch head (`HEAD`) to `<commit>`, optionally modifying index and working tree to match. The `<tree-ish>`/`<commit>` defaults to `HEAD` in all forms.

git reset [-q] [<tree-ish>] [--] <pathspec>…
git reset [-q] [--pathspec-from-file=<file> [--pathspec-file-nul]] [<tree-ish>]

These forms reset the index entries for all paths that match the `<pathspec>` to their state at `<tree-ish>`. (It does not affect the working tree or the current branch.)

This means that `git reset <pathspec>` is the opposite of `git add <pathspec>`. This command is equivalent to `git restore [--source=<tree-ish>] --staged <pathspec>...`.

After running `git reset <pathspec>` to update the index entry, you can use git-restore to check the contents out of the index to the working tree. Alternatively, using git-restore and specifying a commit with `--source`, you can copy the contents of a path out of a commit to the index and to the working tree in one go.

git reset (--patch | -p) [<tree-ish>] [--] [<pathspec>…]

Interactively select hunks in the difference between the index and `<tree-ish>` (defaults to

HEAD). The chosen hunks are applied in reverse to the index.

This means that `git reset -p` is the opposite of `git add -p`, i.e. you can use it to selectively reset hunks. See the “Interactive Mode” section of git-add to learn how to operate the `--patch` mode.

git reset [<mode>] [<commit>]

This form resets the current branch head to `<commit>` and possibly updates the index (resetting it to the tree of `<commit>`) and the working tree depending on `<mode>`. Before the operation, `ORIG_HEAD` is set to the tip of the current branch. If `<mode>` is omitted, defaults to `--mixed`. The `<mode>` must be one of the following:

--soft

Does not touch the index file or the working tree at all (but resets the head to `<commit>`, just like all modes do). This leaves all your changed files "Changes to be committed", as `git status` would put it.

--mixed

Resets the index but not the working tree (i.e., the changed files are preserved but not marked for commit) and reports what has not been updated. This is the default action.

If `-N` is specified, removed paths are marked as intent-to-add (see git-add[1]).

--hard

Resets the index and working tree. Any changes to tracked files in the working tree since `<commit>` are discarded. Any untracked files or directories in the way of writing any tracked files are simply deleted.

--merge

Resets the index and updates the files in the working tree that are different between `<commit>` and `HEAD`, but keeps those which are different between the index and working tree (i.e. which have changes which have not been added). If a file that is different between `<commit>` and the index has unstaged changes, reset is aborted.

In other words, `--merge` does something like a `git read-tree -u -m <commit>`, but carries forward unmerged index entries.

--keep

Resets index entries and updates files in the working tree that are different between `<commit>` and `HEAD`. If a file that is different between `<commit>` and `HEAD` has local changes, reset is aborted.

--[no-]recurse-submodules

When the working tree is updated, using --recurse-submodules will also recursively reset the working tree of all active submodules according to the commit recorded in the superproject, also setting the submodules' HEAD to be detached at that commit.

NAME

git-rm - Remove files from the working tree and from the index

DESCRIPTION

Remove files matching pathspec from the index, or from the working tree and the index. git rm will not remove a file from just your working directory. (There is no option to remove a file only from the working tree and yet keep it in the index; use /bin/rm if you want to do that.) The files being removed have to be identical to the tip of the branch, and no updates to their contents can be staged in the index, though that default behavior can be overridden with the -f option. When --cached is given, the staged content has to match either the tip of the branch or the file on disk, allowing the file to be removed from just the index. When sparse-checkouts are in use (see git-sparse-checkout[1]), git rm will only remove paths within the sparse-checkout patterns.

NAME

git-log - Show commit logs

DESCRIPTION

Shows the commit logs.

List commits that are reachable by following the parent links from the given commit(s), but exclude commits that are reachable from the one(s) given with a ^ in front of them. The output is given in reverse chronological order by default.

You can think of this as a set operation. Commits reachable from any of the commits given on the command line form a set, and then commits reachable from any of the ones given with ^ in front are subtracted from that set. The remaining commits are what comes out in the command's output. Various other options and paths parameters can be used to further limit the result.

Thus, the following command:

$ git log foo bar ^baz

means "list all the commits which are reachable from foo or bar, but not from baz".

A special notation "<commit1>..<commit2>" can be used as a short-hand for "^<commit1> <commit2>". For example, either of the following may be used interchangeably:

$ git log origin..HEAD
$ git log HEAD ^origin

Another special notation is "<commit1>…<commit2>" which is useful for merges. The resulting set of commits is the symmetric difference between the two operands. The following two commands are equivalent:

$ git log A B --not $(git merge-base --all A B)
$ git log A...B

The command takes options applicable to the git-rev-list[1] command to control what is shown and how, and options applicable to the git-diff[1] command to control how the changes each commit introduces are shown.

NAME

git-status - Show the working tree status

DESCRIPTION

Displays paths that have differences between the index file and the current HEAD commit, paths that have differences between the working tree and the index file, and paths in the working tree that are not tracked by Git (and are not ignored by gitignore[5]). The first are what you would commit by running git commit; the second and third are what you could commit by running git add before running git commit.

NAME

git-checkout - Switch branches or restore working tree files

DESCRIPTION

Updates files in the working tree to match the version in the index or the specified tree. If no pathspec was given, git checkout will also update HEAD to set the specified branch as the current branch.

git checkout [<branch>]

To prepare for working on <branch>, switch to it by updating the index and the files in the working tree, and by pointing HEAD at the branch. Local modifications to the files in the working tree are kept, so that they can be committed to the <branch>.

If <branch> is not found but there does exist a tracking branch in exactly one remote (call it <remote>) with a matching name and --no-guess is not specified, treat as equivalent to

$ git checkout -b <branch> --track <remote>/<branch>

You could omit <branch>, in which case the command degenerates to "check out the current branch", which is a glorified no-op with rather expensive side-effects to show only the tracking information, if exists, for the current branch.
git checkout -b|-B <new-branch> [<start-point>]

Specifying -b causes a new branch to be created as if git-branch[1] were called and then checked out. In this case you can use the --track or --no-track options, which will be passed to git branch. As a convenience, --track without -b implies branch creation; see the description of --track below.

If -B is given, <new-branch> is created if it doesn’t exist; otherwise, it is reset. This is the transactional equivalent of

$ git branch -f <branch> [<start-point>]
$ git checkout <branch>

that is to say, the branch is not reset/created unless "git checkout" is successful.
git checkout --detach [<branch>]
git checkout [--detach] <commit>

Prepare to work on top of <commit>, by detaching HEAD at it (see "DETACHED HEAD" section), and updating the index and the files in the working tree. Local modifications to the files in the working tree are kept, so that the resulting working tree will be the state recorded in the commit plus the local modifications.

When the <commit> argument is a branch name, the --detach option can be used to detach HEAD at the tip of the branch (git checkout <branch> would check out that branch without detaching HEAD).

Omitting <branch> detaches HEAD at the tip of the current branch.
git checkout [-f|--ours|--theirs|-m|--conflict=<style>] [<tree-ish>] [--] <pathspec>…
git checkout [-f|--ours|--theirs|-m|--conflict=<style>] [<tree-ish>] --pathspec-from-file=<file> [--pathspec-file-nul]

Overwrite the contents of the files that match the pathspec. When the <tree-ish> (most often a commit) is not given, overwrite working tree with the contents in the index. When the <tree-ish> is given, overwrite both the index and the working tree with the contents at the <tree-ish>.

The index may contain unmerged entries because of a previous failed merge. By default, if you try to check out such an entry from the index, the checkout operation will fail and nothing will be checked out. Using -f will ignore these unmerged entries. The contents from a specific side of the merge can be checked out of the index by using --ours or --theirs. With -m, changes made to the working tree file can be discarded to re-create the original conflicted merge result.
git checkout (-p|--patch) [<tree-ish>] [--] [<pathspec>…]

This is similar to the previous mode, but lets you use the interactive interface to show the "diff" output and choose which hunks to use in the result. See below for the description of --patch option.

NAME

git-commit - Record changes to the repository

DESCRIPTION

Create a new commit containing the current contents of the index and the given log message describing the changes. The new commit is a direct child of HEAD, usually the tip of the current branch, and the branch is updated to point to it (unless no branch is associated with the working tree, in which case HEAD is "detached" as described in

git-checkout[1]).

The content to be committed can be specified in several ways:

by using git-add[1] to incrementally "add" changes to the index before using the commit command (Note: even modified files must be "added");

by using git-rm[1] to remove files from the working tree and the index, again before using the commit command;

by listing files as arguments to the commit command (without --interactive or --patch switch), in which case the commit will ignore changes staged in the index, and instead record the current content of the listed files (which must already be known to Git);

by using the -a switch with the commit command to automatically "add" changes from all known files (i.e. all files that are already listed in the index) and to automatically "rm" files in the index that have been removed from the working tree, and then perform the actual commit;

by using the --interactive or --patch switches with the commit command to decide one by one which files or hunks should be part of the commit in addition to contents in the index, before finalizing the operation. See the “Interactive Mode” section of git-add[1] to learn how to operate these modes.

The --dry-run option can be used to obtain a summary of what is included by any of the above for the next commit by giving the same set of parameters (options and paths).

If you make a commit and then find a mistake immediately after that, you can recover from it with git reset.

7 Configuració inicial

user.name=Javier Garcia Vera

user.email=cf22javier.garcia@iesjoandaustria.org

core.repositoryformatversion=0

core.filemode=true

core.bare=false

core.logallrefupdates=true

remote.origin.url=git@github.com:Javiergave/introM3

remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*

branch.main.remote=origin

branch.main.merge=refs/heads/main

pull.rebase=false

Un repository *bare* és un repository on no es pot escriure codi, sino creat per a compartir-ho sense edició

8 Resum de comandes

git-clone - Clone a repository into a new directory

git-init - Create an empty Git repository or reinitialize an existing one

git-add - Add file contents to the index

git-mv - Move or rename a file, a directory, or a symlink

git-reset - Reset current HEAD to the specified state

git-rm - Remove files from the working tree and from the index

git-log - Show commit logs

git-status - Show the working tree status

git-checkout - Switch branches or restore working tree files

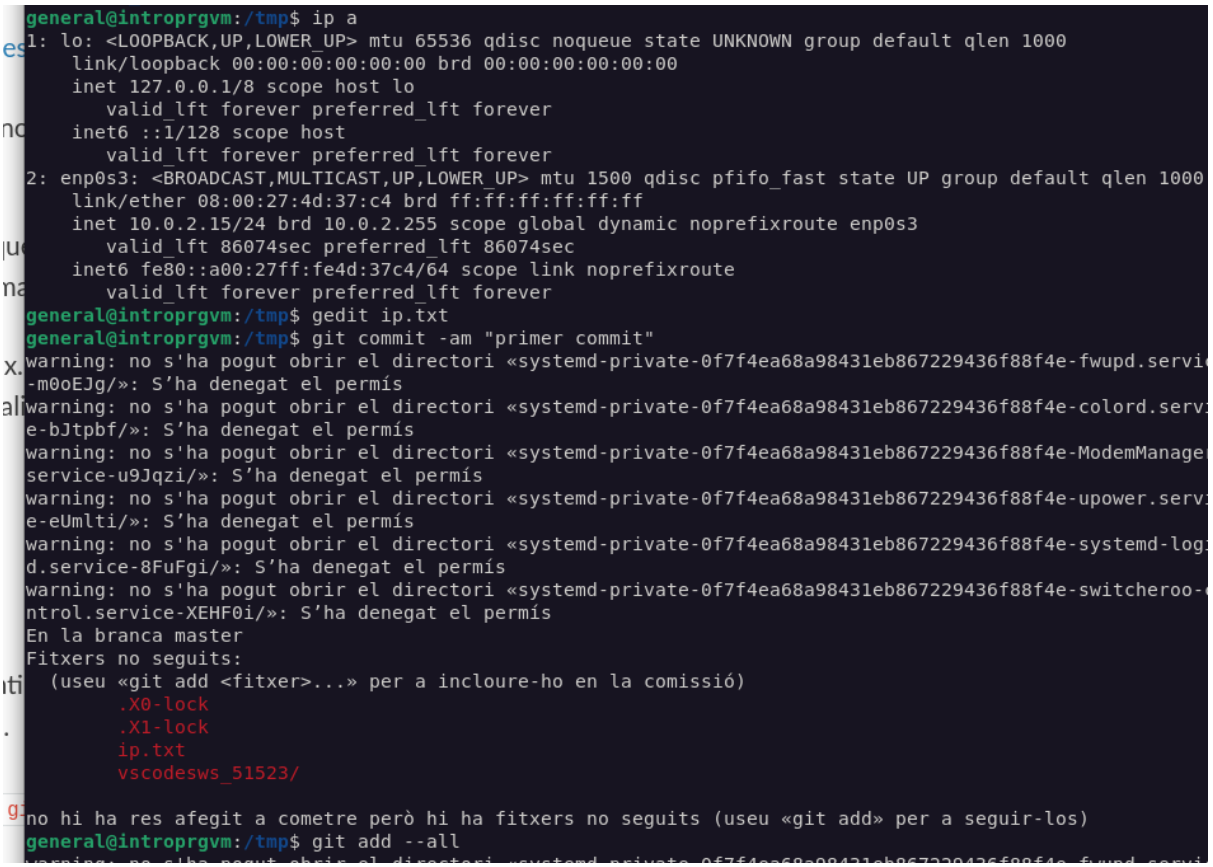
git-commit - Record changes to the repository

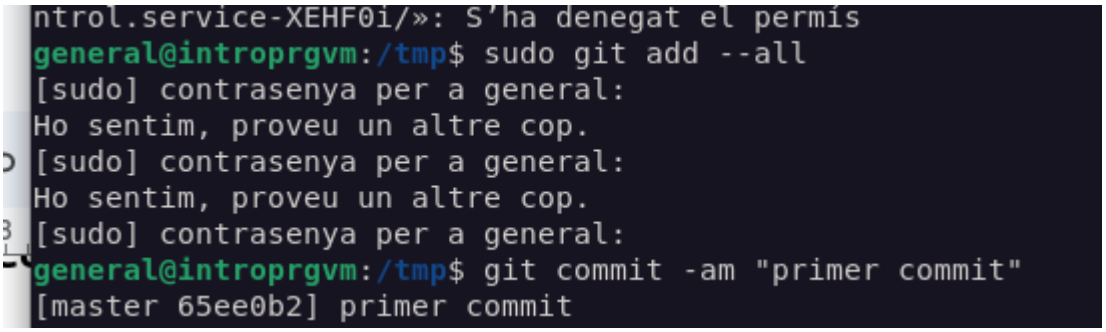
9 Comptem objectes

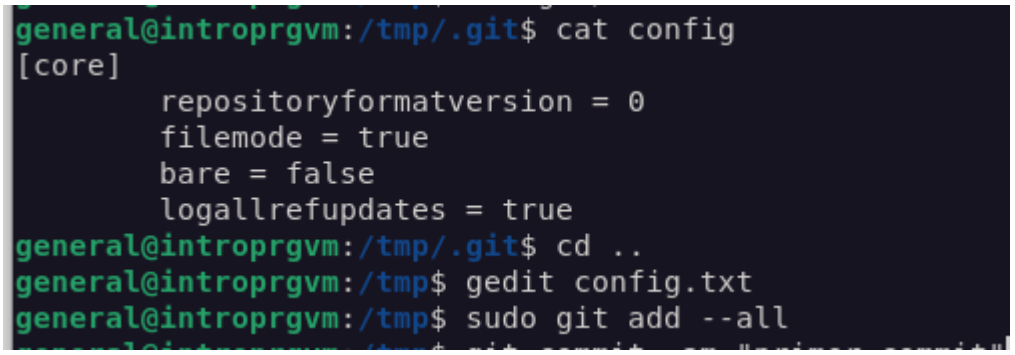
```
general@introprgvm:/tmp$ git count-objects
0 objects, 0 kilobytes
general@introprgvm:/tmp$ gedit test.txt
general@introprgvm:/tmp$ git count-objects
0 objects, 0 kilobytes
general@introprgvm:/tmp$ git add test.txt
general@introprgvm:/tmp$ git count-objects
1 objects, 4 kilobytes
general@introprgvm:/tmp$ git commit -am "a"
[master (comissió arrel) daebff7] a
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
general@introprgvm:/tmp$ git count-objects
3 objects, 12 kilobytes
general@introprgvm:/tmp$ git config --list > test.txt
general@introprgvm:/tmp$ git count-objects
3 objects, 12 kilobytes
general@introprgvm:/tmp$ git commit -am "a"
[master 15aaaeb] a
 1 file changed, 6 insertions(+), 1 deletion(-)
general@introprgvm:/tmp$ git count-objects
6 objects, 24 kilobytes
```

pas	objectes	kilobytes
0	0	0
1	0	0
2	1	4
3	3	12
4	3	12
5	6	24

10 Una mica de pràctica





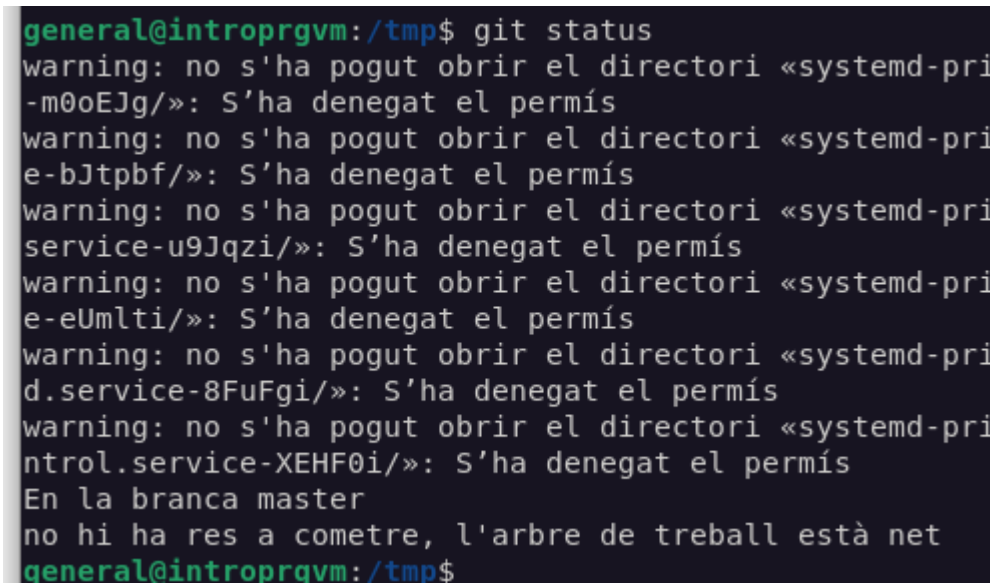


```
general@intropgvm:/tmp$ date
dijous, 27 d'abril de 2023, 18:33:46 CEST
general@intropgvm:/tmp$ gedit date.txt
general@intropgvm:/tmp$ git stage date.txt
general@intropgvm:/tmp$
```

```
general@intropgvm:/tmp$ git stage date.txt
general@intropgvm:/tmp$ git status
warning: no s'ha pogut obrir el directori «systemd-private-0f7f4ea68a98431eb867229436f88f4e-
-m0oEJg/»: S'ha denegat el permís
warning: no s'ha pogut obrir el directori «systemd-private-0f7f4ea68a98431eb867229436f88f4e-
e-bJtpbf/»: S'ha denegat el permís
warning: no s'ha pogut obrir el directori «systemd-private-0f7f4ea68a98431eb867229436f88f4e-
service-u9Jqzi/»: S'ha denegat el permís
warning: no s'ha pogut obrir el directori «systemd-private-0f7f4ea68a98431eb867229436f88f4e-
e-eUmlti/»: S'ha denegat el permís
warning: no s'ha pogut obrir el directori «systemd-private-0f7f4ea68a98431eb867229436f88f4e-
d.service-8FuFgi/»: S'ha denegat el permís
warning: no s'ha pogut obrir el directori «systemd-private-0f7f4ea68a98431eb867229436f88f4e-
ntrol.service-XEHF0i/»: S'ha denegat el permís
En la branca master
Canvis a cometre:
  (use "git restore --staged <file>..." to unstage)
        fitxer nou:       config.txt
        fitxer nou:       date.txt
```

```
general@intropgvm:/tmp$ gedit config.txt
general@intropgvm:/tmp$ git status
warning: no s'ha pogut obrir el directori «systemd-private-0f
-m0oEJg/»: S'ha denegat el permís
warning: no s'ha pogut obrir el directori «systemd-private-0f
e-bJtpbf/»: S'ha denegat el permís
warning: no s'ha pogut obrir el directori «systemd-private-0f
service-u9Jqzi/»: S'ha denegat el permís
warning: no s'ha pogut obrir el directori «systemd-private-0f
e-eUmlti/»: S'ha denegat el permís
warning: no s'ha pogut obrir el directori «systemd-private-0f
d.service-8FuFgi/»: S'ha denegat el permís
warning: no s'ha pogut obrir el directori «systemd-private-0f
ntrol.service-XEHF0i/»: S'ha denegat el permís
En la branca master
Canvis a cometre:
  (use "git restore --staged <file>..." to unstage)
        fitxer nou:       config.txt
        fitxer nou:       date.txt

Canvis no «staged» per a cometre:
  (useu «git add <fitxer>...» per a actualitzar què es cometr
  (use "git restore <file>..." to discard changes in working
        modificat:        config.txt
```



11 Visualització

12 L'art de les comandes