

Control de versions

Taula de continguts

Pregunta 1. La meva història de por.....	1
Pregunta 2. Comparem sistemes de control de versions.....	2
Pregunta 3. Quantes versions guardem?.....	4
Pregunta 4. Configuració global.....	5
Pregunta 5. Ajut d'algunes comandes interessants.....	6
Pregunta 6. Configuració inicial.....	12
Pregunta 7. Resum de comandes.....	13
Pregunta 8. Comptem objectes.....	15
Pregunta 9. Una mica de pràctica.....	18
Pregunta 10. Visualització.....	23
Pregunta 11. L'art de la línia de comandes.....	24

Introducció

L'objectiu d'aquest exercici és que coneguem i aprenem sobre el control de versions i més específicament el git. Gràcies al control de versions podem modificar el nostre codi sense tenir por de no poder desfer-ho ja que tenim una còpia de seguretat on recolzar-nos.

Tenim 11 seccions en les quals practiquem el funcionament i entenem millor els controls de versions.

A mesura que fem seccions aprenem més sobre el funcionament del git. La primera secció és molt teòrica i poc a poc anem aplicant la teoria a la pràctica fins arribar a les últimes seccions on el que hem de fer és aplicar tot el coneixement que hem adquirit a les primeres.

Pregunta 1. La meva història de por

- **Recuperacions**

- Quan estava fent l'exercici 31_40 vaig esborrar una part per provar una idea que vaig tenir. La idea no era bona i no em va arreglar l'error, quan vaig intentar retornar a la versió antiga ja s'havia perdut. No vaig poder recuperar-la i vaig haver de fer-ho tot un altre cop. Ara em sembla molt fàcil però quan em va passar em va semblar una muntanya. Això m'ha passat massa cops ja que no sé com assegurar-me de que la versió antiga segueix disponible.

Pregunta 2. Comparem sistemes de control de versions

- SCV centralitzats

1. CVS

Nom del CVS: **Concurrent Versions System (CVS)**

URL del projecte: <http://www.nongnu.org/cvs/>

Llicència: GNU General **Public License (GPL)**

Descripció: **CVS** és un sistema de control de versions centralitzat dissenyat per a permetre als desenvolupadors col·laborar en un projecte de programari de manera eficient. Proporciona funcions per a la gestió de versions i control de canvis en els arxius del projecte.

Bondats:

Permet la gestió de diferents versions d'un arxiu i la recuperació de versions anteriors.

Facilita la col·laboració i treball en equip en permetre als desenvolupadors treballar en la mateixa versió del codi font.

És fàcil d'usar i configurar.

És compatible amb una gran quantitat d'eines de programari.

Impressió: **CVS** és una opció sòlida per a la gestió de versions de programari, però el seu disseny centralitzat pot presentar desavantatges en termes de disponibilitat del servidor i escalabilitat.

2. Subversion

Nom del CVS: Apatxe **Subversion (SVN)**

URL del projecte: <https://subversion.apache.org/>

Llicència: Apatxe **License, Version 2.0**

Descripció: **SVN** és un sistema de control de versions centralitzat que permet la gestió i control de canvis en els arxius d'un projecte. Està dissenyat per a ser una alternativa millorada i més segura que **CVS**.

Bondats:

Proporciona funcions de control de versions i control de canvis en els arxius d'un projecte.

És fàcil d'aprendre i utilitzar.

Suporta una gran quantitat de característiques avançades, com la fusió de branques i la creació d'etiquetes.

Ofereix una **API** per a desenvolupadors que permet integrar **SVN** amb altres eines de programari.

Impressió: **SVN** és una opció sòlida per a la gestió de versions de programari centralitzat, però ha estat reemplaçat en popularitat per sistemes de control de versions distribuïts com **Git**.

- SCV distribuïts

3. Git

Nom del CVS: Git

URL del projecte: <https://git-scm.com/>

Llicència: GNU General Public License (GPL)

Descripció: Git és un sistema de control de versions distribuït dissenyat per a manejar projectes de programari de qualsevol grandària amb velocitat i eficiència. Proporciona una àmplia gamma de funcions per a la gestió de versions i control de canvis en els arxius d'un projecte.

Bondats:

És ràpid, eficient i escalable.

Proporciona una gran quantitat de funcions avançades, com la creació de branques i etiquetes, i la fusió de branques.

Permet la col·laboració i treball en equip en temps real sense la necessitat d'un servidor centralitzat.

És compatible amb una gran quantitat d'eines de programari.

Impressió: Git és una opció molt popular i sòlida per a la gestió de versions de programari. El seu disseny distribuït ho fa més resistent a falles i problemes d'escalabilitat que els sistemes centralitzats com SVN.

4. Mercurial

Nom del CVS: Mercurial

URL del projecte: <https://www.mercurial-scm.org/>

Llicència: GNU General Public License (GPL)

Descripció: Mercurial és un sistema de control de versions distribuït que se centra en la velocitat, l'escalabilitat i la facilitat d'ús. Proporciona funcions per a la gestió de versions i control de canvis en els arxius d'un projecte.

Bondats:

És ràpid, eficient i escalable.

Proporciona una interfície d'usuari senzilla i fàcil d'usar.

Permet la col·laboració i treball en equip en temps real sense la necessitat d'un servidor centralitzat.

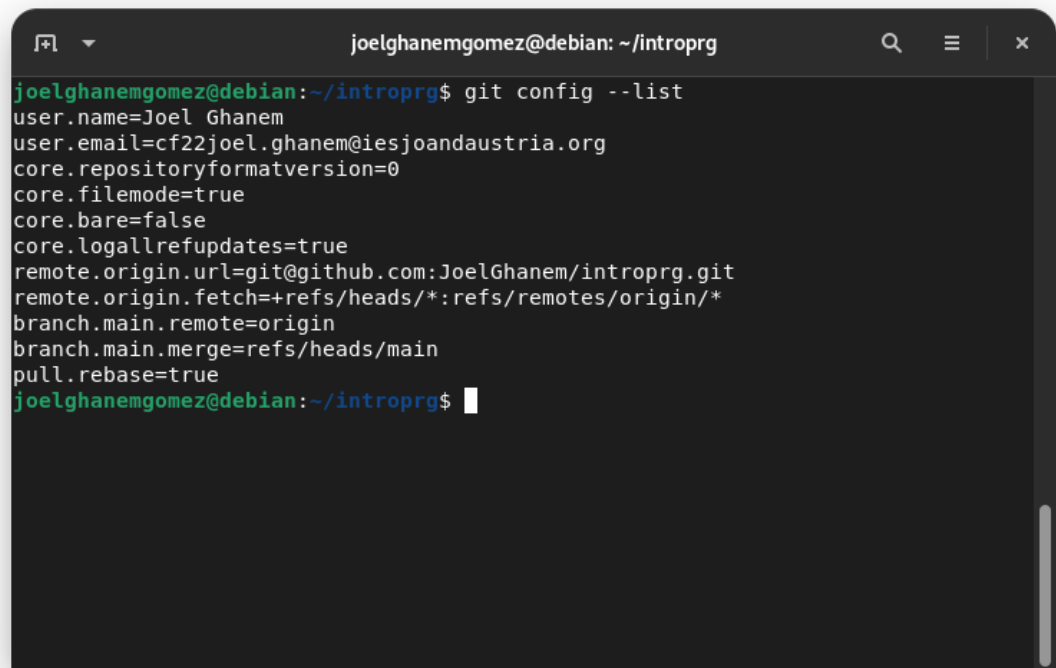
Ofereix una gran quantitat de característiques avançades, com la creació de branques i etiquetes, i la fusió de branques.

Impressió: Mercurial és una opció sòlida per a la gestió de versions de programari distribuït. El seu enfocament en la facilitat d'ús i la seva interfície d'usuari senzilla ho fan atractiu per als desenvolupadors que busquen una alternativa més simple que Git.

Pregunta 3. Quantes versions guardem?

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

Pregunta 4. Configuració global

A terminal window with a dark background. The title bar shows 'joelghanemgomez@debian: ~/introprg'. The prompt is 'joelghanemgomez@debian:~/introprg\$'. The command 'git config --list' has been executed, and the output is displayed in a monospaced font. The output lists various git configuration settings, including user information, core settings, and remote configurations. The prompt is now 'joelghanemgomez@debian:~/introprg\$' with a cursor.

```
joelghanemgomez@debian:~/introprg$ git config --list
user.name=Joel Ghanem
user.email=cf22joel.ghanem@iesjoandaustria.org
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=git@github.com:JoelGhanem/introprg.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.main.remote=origin
branch.main.merge=refs/heads/main
pull.rebase=true
joelghanemgomez@debian:~/introprg$
```


Pregunta 5. Ajut d'algunes comandes interessants

*clone

```
joelghanemgomez@debian: ~  
GIT-CLONE(1)      Git Manual      GIT-CLONE(1)  
  
NAME  
    git-clone - Clone a repository into a new directory  
  
SYNOPSIS  
    git clone [--template=<template_directory>]  
              [-l] [-s] [--no-hardlinks] [-q] [-n] [--bare] [--mirror]  
              [-o <name>] [-b <name>] [-u <upload-pack>] [--reference <repository>]  
              [--dissociate] [--separate-git-dir <git dir>]  
              [--depth <depth>] [--[no-]single-branch] [--no-tags]  
              [--recurse-submodules[=<paths-spec>]] [--[no-]shallow-submodules]  
              [--[no-]remote-submodules] [--jobs <n>] [--sparse]  
              [--filter=<filter>] [--] <repository>  
              <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.
```

*init

```
joelghanemgomez@debian: ~  
GIT-INIT(1)      Git Manual      GIT-INIT(1)  
  
NAME  
    git-init - Create an empty Git repository or reinitialize an existing one  
  
SYNOPSIS  
    git init [-q | --quiet] [--bare] [--template=<template_directory>]  
              [--separate-git-dir <git dir>] [--object-format=<format>]  
              [-b <branch-name> | --initial-branch=<branch-name>]  
              [--shared[=<permissions>]] [directory]  
  
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 shl 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).
```

*add

```
joelghanemgomez@debian: ~  
GIT-ADD(1)                               Git Manual                               GIT-ADD(1)  
  
NAME  
    git-add - Add file contents to the index  
  
SYNOPSIS  
    git add [--verbose | -v] [--dry-run | -n] [--force | -f] [--interactive | -i] [--patch | -p]  
            [--edit | -e] [--[no-]all | --[no-]ignore-removal | [--update | -u]  
            [--intent-to-add | -N] [--refresh] [--ignore-errors] [--ignore-missing] [--renormalize]  
            [--chmod=(+|-)x] [--pathspec-from-file=<file>] [--pathspec-file-nul]  
            [--] [<pathspec>...]  
  
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.  
  
    Please see git-commit(1) for alternative ways to add content to a commit.
```

*mv

```
joelghanemgomez@debian: ~  
GIT-MV(1)                               Git Manual                               GIT-MV(1)  
  
NAME  
    git-mv - Move or rename a file, a directory, or a symlink  
  
SYNOPSIS  
    git mv <options>... <args>...  
  
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.
```

* reset

```
joelghanemgomez@debian: ~
GIT-RESET(1)                               Git Manual                               GIT-RESET(1)

NAME
    git-reset - Reset current HEAD to the specified state

SYNOPSIS
    git reset [-q] [<tree-ish>] [--] [<pathspec>...]
    git reset [-q] [--pathspec-from-file=<file> [--pathspec-file-nul]] [<tree-ish>]
    git reset [--patch | -p] [<tree-ish>] [--] [<pathspec>...]
    git reset [--soft | --mixed [-N] | --hard | --merge | --keep] [-q] [<commit>]

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(1)
        to check the contents out of the index to the working tree. Alternatively, using git-
        restore(1) 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(1) 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>. 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.

        --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.

        See "Reset, restore and revert" in git(1) for the differences between the three commands.

OPTIONS
    Manual page git-reset(1) line 25 (press h for help or q to quit)
```

*rm

```
joelghanemgomez@debian: ~  
GIT-RM(1) Git Manual GIT-RM(1)  
  
NAME  
    git-rm - Remove files from the working tree and from the index  
  
SYNOPSIS  
    git rm [-f | --force] [-n] [-r] [--cached] [--ignore-unmatch]  
           [--quiet] [--pathspec-from-file=<file> [--pathspec-file-nul]]  
           [--] [<pathspec>...]  
  
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.
```

*log

```
joelghanemgomez@debian: ~  
GIT-LOG(1) Git Manual GIT-LOG(1)  
  
NAME  
    git-log - Show commit logs  
  
SYNOPSIS  
    git log [<options>] [<revision range>] [--] [<path>...]  
  
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.
```

*status

```
joelghanemgomez@debian: ~  
GIT-STATUS(1)                               Git Manual                               GIT-STATUS(1)  
  
NAME  
    git-status - Show the working tree status  
  
SYNOPSIS  
    git status [<options>...] [--] [<pathspec>...]  
  
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.
```

*checkout

```
joelghanemgomez@debian: ~  
GIT-CHECKOUT(1)                             Git Manual                             GIT-CHECKOUT(1)  
  
NAME  
    git-checkout - Switch branches or restore working tree files  
  
SYNOPSIS  
    git checkout [-q] [-f] [-m] [<branch>]  
    git checkout [-q] [-f] [-m] --detach [<branch>]  
    git checkout [-q] [-f] [-m] [--detach] <commit>  
    git checkout [-q] [-f] [-m] [--b|-B|--orphan] <new branch> [<start point>]  
    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>  
    git checkout [-f|--ours|--theirs|-m|--conflict=<style>] [<tree-ish>] --pathspec-from-file-nul  
    git checkout (-p|--patch) [<tree-ish>] [--] [<pathspec>...]  
  
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  
Manual page git-checkout(1) line 1 (press h for help or q to quit)
```

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.

Manual page git-checkout(1) line 24 (press h for help or q to quit)

*commit

```
joelghanemgomez@debian: ~
GIT-COMMIT(1)          Git Manual          GIT-COMMIT(1)

NAME
  git-commit - Record changes to the repository

SYNOPSIS
  git commit [-a | --interactive | --patch] [-s] [-v] [-u<mode>] [--amend]
              [--dry-run] [(--c | -C | --fixup | --squash) <commit>]
              [-F <file> | -m <msg>] [--reset-author] [--allow-empty]
              [--allow-empty-message] [--no-verify] [-e] [--author=<author>]
              [--date=<date>] [--cleanup=<mode>] [--no-]status
              [-i | -o] [--pathspect-from-file=<file>] [--pathspect-file-nul]
              [-S<keyid>] [--] [<pathspec>...]

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:

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

  2. by using git-rm(1) to remove files from the working tree and the index, again before using
     the commit command;

  3. 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);

  4. 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;

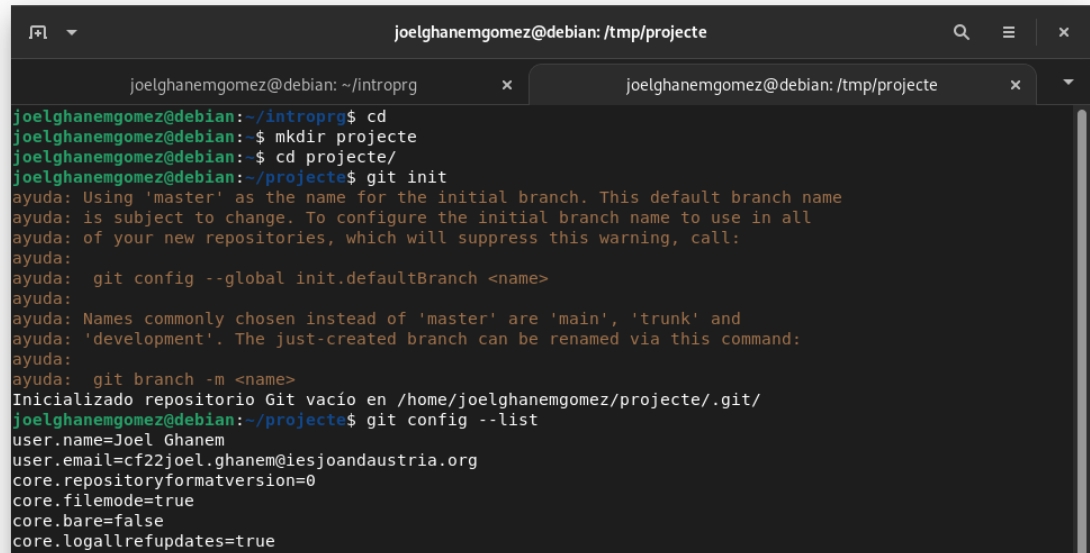
  5. by using the --interactive or --patch switches with the commit command to decide one by
     one which files or hunks should be part of the commit in addition to contents in the
     index, before finalizing the operation. See the "Interactive Mode" section of git-add(1)
     to learn how to operate 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.
```

Pregunta 6. Configuració inicial

1.



```
joelghanemgomez@debian: /tmp/projecte
joelghanemgomez@debian: ~/introprg
joelghanemgomez@debian:~$ cd
joelghanemgomez@debian:~$ mkdir projecte
joelghanemgomez@debian:~$ cd projecte/
joelghanemgomez@debian:~/projecte$ git init
ayuda: Using 'master' as the name for the initial branch. This default branch name
ayuda: is subject to change. To configure the initial branch name to use in all
ayuda: of your new repositories, which will suppress this warning, call:
ayuda: git config --global init.defaultBranch <name>
ayuda: Names commonly chosen instead of 'master' are 'main', 'trunk' and
ayuda: 'development'. The just-created branch can be renamed via this command:
ayuda: git branch -m <name>
Iniciado repositorio Git vacío en /home/joelghanemgomez/projecte/.git/
joelghanemgomez@debian:~/projecte$ git config --list
user.name=Joel Ghanem
user.email=cf22joel.ghanem@iesjoandaustria.org
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
```

2.

Voldrà dir que el nucli del nou repository no es bare ja que tenim un directori previ

Pregunta 7. Resum de comandes

git init: Crea un nou repositori Git al directori actual git init nouRepositori

<code>git init</code>	Crea un nou repositori Git al directori actual.	<code>\$ git init nouRep</code>
<code>git add</code> <code><nom_fitxer></code>	Afegir un fitxer específic al stage.	<code>\$ git add file.txt</code>
<code>git add --all</code>	Afegir tots els fitxers nous o modificats al stage.	<code>\$ git add -all</code>
<code>git commit -am</code> <code>"comentaris"</code>	Registra els canvis a stage associant-los un comentari.	<code>\$ git commit -m</code> <code>"Afegit contingut al fitxer"</code>
<code>git log</code>	Mostra l'historial de commits realitzats en el repositori.	<code>\$ git log --oneline</code>
<code>git diff</code>	Mostra les diferències entre el directori de treball i el stage.	<code>\$ git diff file.txt</code>
<code>git diff --staged</code>	Mostra les diferències entre el stage i el darrer commit realitzat.	<code>\$ git diff --staged file.txt</code>
<code>git checkout --</code> <code><nom_fitxer></code>	Torna al contingut del darrer commit per al fitxer especificat.	<code>\$ git checkout -- file.txt</code>
<code>git branch</code> <code><nom_branca></code>	Crea una nova branca amb el nom especificat.	<code>\$ git branch new-branch</code>
<code>git checkout</code> <code><nom_branca></code>	Canvia a la branca especificada.	<code>\$ git checkout main</code>
<code>git merge</code> <code><nom_branca></code>	Fusiona la branca especificada a la branca actual.	<code>\$ git merge novaBranca</code>
<code>git branch -d</code> <code><nom_branca></code>	Elimina la branca especificada.	<code>\$ git branch -d novaBranca</code>
<code>git push</code>	Puja els commits locals a un repositori remot.	<code>\$ git push</code>

<code>git pull</code>	Descarrega els commits d'un repositori remot i els fusiona amb la branca local.	<code>\$ git pull</code>
<code>git reset --hard <commit></code>	Torna al commit especificat i descarta qualsevol canvi posterior. Aquesta comanda és perillosa i es recomana fer-ne ús amb precaució.	<code>\$ git reset --hard abc123</code>
<code>git reset --soft <commit></code>	Torna al commit especificat, però manté els canvis posteriors com a canvis no afegits.	<code>\$ git reset --soft abc123</code>
<code>git cherry-pick <commit></code>	Aplica els canvis d'un commit específic a la branca actual.	<code>\$ git cherry-pick abc123</code>
<code>git stash</code>	Emmagatzema els canvis actuals sense fer commit, per poder treballar en una branca diferent.	<code>\$ git stash save "Canvis a fer en la nova branca"</code>
<code>git stash apply</code>	Restaura els canvis emmagatzemats amb la comanda "git stash".	<code>\$ git stash apply stash@{0}</code>
<code>git stash drop</code>	Elimina els canvis emmagatzemats amb la comanda "git stash".	<code>\$ git stash drop stash@{0}</code>
<code>git stash pop</code>	Restaura i elimina els canvis emmagatzemats amb la comanda "git stash".	<code>\$ git stash pop</code>
<code>git remote add <nom_repositori> <url></code>	Afegeix un repositori remot amb el nom especificat i la URL especificada.	<code>\$ git remote add origin https://github.com/JoelGhanem/introprg.git</code>
<code>git clone <url></code>	Descarrega un repositori remot a la carpeta actual.	<code>\$ git clone https://github.com/JoelGhanem/introprg.git</code>
<code>git push -u</code>	Puja la branca especificada a un	<code>\$ git push -u</code>

<code><nom_repositori></code> <code><branca></code>	repositori remot amb el nom especificat, establint-la com a branca per defecte en futures pujades.	<code>origin main</code>
<code>git status</code>	Mostra l'estat actual del directori de treball i el stage.	<code>\$ git status</code>
<code>git config --global user.name "<nom>"</code>	Configura el nom d'usuari que s'utilitzarà en els commits del repositori.	<code>\$ git config --global user.name "JoelGhanem"</code>
<code>git config --global user.email "<email>"</code>	Configura l'adreça de correu electrònic que s'utilitzarà en els commits del repositori.	<code>\$ git config --global user.email "cf22joel.ghanem@ie sjoandaustria.org"</code>

Pregunta 8. Comptem objectes

```
joelghanemgomez@debian: /tmp/projecte
joelghanemgomez@debian: ~/introprg x joelghanemgomez@debian: /tmp/projecte x
joelghanemgomez@debian:~$ cd -tmp
bash: cd: -t: opción inválida
cd: modo de empleo: cd [-L|[-P [-e]] [dir]
joelghanemgomez@debian:~$ cd /tmp
joelghanemgomez@debian:/tmp$ mkdir projecte
joelghanemgomez@debian:/tmp$ cd projecte/
joelghanemgomez@debian:/tmp/projecte$ git init
ayuda: Using 'master' as the name for the initial branch. This default branch name
ayuda: is subject to change. To configure the initial branch name to use in all
ayuda: of your new repositories, which will suppress this warning, call:
ayuda: git config --global init.defaultBranch <name>
ayuda:
ayuda: Names commonly chosen instead of 'master' are 'main', 'trunk' and
ayuda: 'development'. The just-created branch can be renamed via this command:
ayuda: git branch -m <name>
Iniciado repositorio Git vacío en /tmp/projecte/.git/
joelghanemgomez@debian:/tmp/projecte$ git count-objects
0 objects, 0 kilobytes
joelghanemgomez@debian:/tmp/projecte$ touch test.txt
joelghanemgomez@debian:/tmp/projecte$ ls
test.txt
joelghanemgomez@debian:/tmp/projecte$ git count-objects
0 objects, 0 kilobytes
joelghanemgomez@debian:/tmp/projecte$ git commit -am "test.txt"
En la rama master

Confirmación inicial

Archivos sin seguimiento:
(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
test.txt

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles
seguimiento)
joelghanemgomez@debian:/tmp/projecte$ git add test.txt
joelghanemgomez@debian:/tmp/projecte$ git count-objects
2 objects, 8 kilobytes
joelghanemgomez@debian:/tmp/projecte$ git commit -am "test.txt"
[master (commit-raíz) 8d6c8bb] test.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test.txt
joelghanemgomez@debian:/tmp/projecte$ git count-objects
4 objects, 16 kilobytes
joelghanemgomez@debian:/tmp/projecte$ git config --list > test.txt
joelghanemgomez@debian:/tmp/projecte$ git count-objects
4 objects, 16 kilobytes
joelghanemgomez@debian:/tmp/projecte$ git commit -am "test.txt"
[master b034c3a] test.txt
1 file changed, 6 insertions(+)
joelghanemgomez@debian:/tmp/projecte$ git count-objects
7 objects, 28 kilobytes
joelghanemgomez@debian:/tmp/projecte$
```

pas	objectes	kilobytes
0	0	0
1	0	0
2	2	8
3	4	16
4	4	16
5	7	28

Pregunta 9. Una mica de pràctica

```
joelghanemgomez@debian: ~/52_01/repositoriNou
joelghanemgomez@deb... x joelghanemgomez@deb... x joelghanemgomez@deb... x joelghanemgomez@deb... x
joelghanemgomez@debian:~/the-art-of-command-line$ cd
joelghanemgomez@debian:~$ cd 52_01/
joelghanemgomez@debian:~/52_01$ ip a
1: lo: <LOOPBACK,UP,LOWER UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp34s0: <BROADCAST,MULTICAST,UP,LOWER UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether d8:bb:c1:10:dc:17 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.81/24 brd 192.168.1.255 scope global dynamic noprefixroute enp34s0
        valid_lft 38290sec preferred_lft 38290sec
    inet6 fe80::dabb:c1ff:fe10:dc17/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
joelghanemgomez@debian:~/52_01$ nvim ipa.txt
joelghanemgomez@debian:~/52_01$ git commit -am "primer commit ipa"
En la rama master

Confirmación inicial

Archivos sin seguimiento:
(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    ipa.txt

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles
seguimiento)
joelghanemgomez@debian:~/52_01$ git add --all
joelghanemgomez@debian:~/52_01$ git commit -am "primer commit ipa"
[master (commit-raíz) 0565236] primer commit ipa
1 file changed, 13 insertions(+)
create mode 100644 ipa.txt
joelghanemgomez@debian:~/52_01$ nvim .git/
branches/      config      HEAD          index          logs/          refs/
COMMIT_EDITMSG description hooks/         info/          objects/
joelghanemgomez@debian:~/52_01$ s
bash: s: orden no encontrada
joelghanemgomez@debian:~/52_01$ ls
ipa.txt
joelghanemgomez@debian:~/52_01$ ls -a
.  .git  ipa.txt
joelghanemgomez@debian:~/52_01$ cd .git/
joelghanemgomez@debian:~/52_01/.git$ ls
branches COMMIT_EDITMSG config description HEAD hooks index info logs objects refs
joelghanemgomez@debian:~/52_01/.git$ ls -a
.  .. branches COMMIT_EDITMSG config description HEAD hooks index info logs objects refs
joelghanemgomez@debian:~/52_01/.git$ cd
joelghanemgomez@debian:~$ cd ..
joelghanemgomez@debian:/home$ ls -a
.  .. joelghanemgomez
joelghanemgomez@debian:/home$ cd joelghanemgomez/
joelghanemgomez@debian:~$ cd ..
joelghanemgomez@debian:/home$ nvim ~/.gitconfig
joelghanemgomez@debian:/home$ cd joelghanemgomez/52_01/
joelghanemgomez@debian:~/52_01$ ls
```

```
joelghanemgomez@debian: ~/52_01/repositoriNou
joelghanemgomez@deb... x joelghanemgomez@deb... x joelghanemgomez@deb... x joelghanemgomez@deb... x
joelghanemgomez@debian:~/52_01$ ls
ipa.txt
joelghanemgomez@debian:~/52_01$ cp ~/.gitconfig gitconfig.txt
joelghanemgomez@debian:~/52_01$ git add gitconfig.txt
joelghanemgomez@debian:~/52_01$ git commit -am "Fitxer de configuracio"
[master 35795d2] Fitxer de configuracio
1 file changed, 3 insertions(+)
create mode 100644 gitconfig.txt
joelghanemgomez@debian:~/52_01$ date > data.txt
joelghanemgomez@debian:~/52_01$ git add data.txt
joelghanemgomez@debian:~/52_01$ git status
En la rama master
Tu rama está basada en 'origin/master', pero upstream ha desaparecido.
(usa "git branch --unset-upstream" para arreglar)

Cambios a ser confirmados:
(usa "git restore --staged <archivo>..." para sacar del área de stage)
nuevo archivo: data.txt

joelghanemgomez@debian:~/52_01$ echo "\n$(date)" >> fitxer.txt
joelghanemgomez@debian:~/52_01$ git status
En la rama master
Tu rama está basada en 'origin/master', pero upstream ha desaparecido.
(usa "git branch --unset-upstream" para arreglar)

Cambios a ser confirmados:
(usa "git restore --staged <archivo>..." para sacar del área de stage)
nuevo archivo: data.txt

Archivos sin seguimiento:
(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
fitxer.txt

joelghanemgomez@debian:~/52_01$ git add --all
joelghanemgomez@debian:~/52_01$ git status
En la rama master
Tu rama está basada en 'origin/master', pero upstream ha desaparecido.
(usa "git branch --unset-upstream" para arreglar)

Cambios a ser confirmados:
(usa "git restore --staged <archivo>..." para sacar del área de stage)
nuevo archivo: data.txt
nuevo archivo: fitxer.txt

joelghanemgomez@debian:~/52_01$ git log --oneline
35795d2 (HEAD -> master) Fitxer de configuracio
0565236 primer commit ipa
joelghanemgomez@debian:~/52_01$ echo "*.class" >> .gitignore
joelghanemgomez@debian:~/52_01$ git add .gitignore
joelghanemgomez@debian:~/52_01$ git status
En la rama master
Tu rama está basada en 'origin/master', pero upstream ha desaparecido.
(usa "git branch --unset-upstream" para arreglar)

Cambios a ser confirmados:
```

```

(usa "git restore --staged <archivo>..." para sacar del área de stage)
nuevo archivo: .gitignore
nuevo archivo: data.txt
nuevo archivo: fitxer.txt

joelghanemgomez@debian:~/52_01$ echo "Canvi" >> fitxer.txt
joelghanemgomez@debian:~/52_01$ git diff
HEAD      master
joelghanemgomez@debian:~/52_01$ git diff --s
--shortstat  --staged      --submodule=  --summary
--src-prefix= --stat       --submodule
joelghanemgomez@debian:~/52_01$ git diff --staged fitxer.txt
diff --git a/fitxer.txt b/fitxer.txt
new file mode 100644
index 0000000..938168c
--- /dev/null
+++ b/fitxer.txt
@@ -0,0 +1 @@
+\n jue 04 may 2023 19:26:05 CEST
joelghanemgomez@debian:~/52_01$ git log -p fitxer.txt
joelghanemgomez@debian:~/52_01$ git checkout -- fitxer.txt
joelghanemgomez@debian:~/52_01$ git branch novaBranca
joelghanemgomez@debian:~/52_01$ git checkout novaBranca
A       .gitignore
A       data.txt
A       fitxer.txt
Cambiado a rama 'novaBranca'
joelghanemgomez@debian:~/52_01$ echo "canvi en novaBranca" >> fitxer.txt
joelghanemgomez@debian:~/52_01$ git add fitxer.txt
joelghanemgomez@debian:~/52_01$ git
add          clone          help          rebase       show-branch
am           commit         init          relog        sparse-checkout
apply        config          instaweb     remote       stage
archive     describe       log           repack       stash
bisect      diff            maintenance  replace      status
blame       difftool       merge         request-pull submodule
branch      fetch          mergetool    reset        switch
bundle      format-patch   mv            restore      tag
checkout    fsck           notes        revert       whatchanged
cherry      gc             prune        rm           worktree
cherry-pick gitk           pull          send-email
citool      grep           push          shortlog
clean       gui            range-diff   show

joelghanemgomez@debian:~/52_01$ git commit -am "canvi"
[novaBranca 17bf03c] canvi
3 files changed, 4 insertions(+)
create mode 100644 .gitignore
create mode 100644 data.txt
create mode 100644 fitxer.txt
joelghanemgomez@debian:~/52_01$ git log --oneline
17bf03c (HEAD -> novaBranca) canvi
35795d2 (master) Fitxer de configuracio
0565236 primer commit ipa
joelghanemgomez@debian:~/52_01$ git checkout master

```

```

Cambiado a rama 'master'
Tu rama está basada en 'origin/master', pero upstream ha desaparecido.
(usa "git branch --unset-upstream" para arreglar)
joelghanemgomez@debian:~/52_01$ git log --oneline
35795d2 (HEAD -> master) Fitxer de configuracio
0565236 primer commit ipa
joelghanemgomez@debian:~/52_01$ git merge novaBranca
Actualizando 35795d2..17bf03c
Fast-forward
 .gitignore | 1 +
 data.txt   | 1 +
 fitxer.txt | 2 ++
 3 files changed, 4 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 data.txt
 create mode 100644 fitxer.txt
joelghanemgomez@debian:~/52_01$ git branch -d novaBranca
Eliminada la rama novaBranca (era 17bf03c)..
joelghanemgomez@debian:~/52_01$ git branch novaBranca2
joelghanemgomez@debian:~/52_01$ git checkout novaBranca2
Cambiado a rama 'novaBranca2'
joelghanemgomez@debian:~/52_01$ echo "canvi novaBranca2" >> fitxer.txt
joelghanemgomez@debian:~/52_01$ git add fitxer.txt
joelghanemgomez@debian:~/52_01$ git commit -am "Canvi en novaBranca2"
[novaBranca2 f077fff] Canvi en novaBranca2
 1 file changed, 1 insertion(+)
joelghanemgomez@debian:~/52_01$ git checkout master
Cambiado a rama 'master'
Tu rama está basada en 'origin/master', pero upstream ha desaparecido.
(usa "git branch --unset-upstream" para arreglar)
joelghanemgomez@debian:~/52_01$ echo "canvi master" >> fitxer.txt
joelghanemgomez@debian:~/52_01$ git add fitxer.txt
joelghanemgomez@debian:~/52_01$ git commit -am "canvi mestre"
[master c787ee2] canvi mestre
 1 file changed, 1 insertion(+)
joelghanemgomez@debian:~/52_01$ git checkout
HEAD      master      novaBranca2  ORIG HEAD
joelghanemgomez@debian:~/52_01$ git checkout novaBranca2
Cambiado a rama 'novaBranca2'
joelghanemgomez@debian:~/52_01$ echo "canvi novaBranca2 que rompe todo" >> fitxer.txt
joelghanemgomez@debian:~/52_01$ git add fitxer.txt
joelghanemgomez@debian:~/52_01$ git commit "canvi error duro"
error: ruta especificada 'canvi error duro' no concordó con ningún archivo(s) conocido por git
joelghanemgomez@debian:~/52_01$ git commit -am "canvi error duro"
[novaBranca2 5be8205] canvi error duro
 1 file changed, 1 insertion(+)
joelghanemgomez@debian:~/52_01$ git checkout master
Cambiado a rama 'master'
Tu rama está basada en 'origin/master', pero upstream ha desaparecido.
(usa "git branch --unset-upstream" para arreglar)
joelghanemgomez@debian:~/52_01$ echo "canvi gordo" >> fitxer.txt
joelghanemgomez@debian:~/52_01$ git add fitxer.txt
joelghanemgomez@debian:~/52_01$ git commit -am "canvi liada" >> fitxer.txt
joelghanemgomez@debian:~/52_01$ git commit -am "canvi liada"

```



```

[master 4a118f8] canvi liada
1 file changed, 2 insertions(+)
joelghanemgomez@debian:~/52_01$ git merge novaBranca2
Auto-fusionando fitxer.txt
CONFLICTO (contenido): Conflicto de fusión en fitxer.txt
Fusión automática falló; arregle los conflictos y luego realice un commit con el resultado.
joelghanemgomez@debian:~/52_01$ git status
En la rama master
Tu rama está basada en 'origin/master', pero upstream ha desaparecido.
(usa "git branch --unset-upstream" para arreglar)

Tienes rutas no fusionadas.
(arregla los conflictos y corre "git commit"
(usa "git merge --abort" para abortar la fusion)

Rutas no fusionadas:
(usa "git add <archivo>..." para marcar una resolución)
    ambos modificados:    fitxer.txt

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
joelghanemgomez@debian:~/52_01$ nvim fitxer.txt
joelghanemgomez@debian:~/52_01$ git add fitxer.txt
joelghanemgomez@debian:~/52_01$ git commit -am "arreglao"
[master 6bca7f8] arreglao
joelghanemgomez@debian:~/52_01$ git merge
merge: refs/remotes/origin/master - nada que podamos fusionar
joelghanemgomez@debian:~/52_01$ git merge novaBranca2
Ya está actualizado.
joelghanemgomez@debian:~/52_01$ git branch -d novaBranca2
Eliminada la rama novaBranca2 (era 5be8205)..
joelghanemgomez@debian:~/52_01$ git bundle create backup.gitbundle master
Enumerando objetos: 29, listo.
Contando objetos: 100% (29/29), listo.
Comprimiendo objetos: 100% (26/26), listo.
Total 29 (delta 13), reusado 0 (delta 0), pack-reusado 0
joelghanemgomez@debian:~/52_01$ git clone backup.gitbundle repositoriNou
Clonando en 'repositoriNou'...
Recibiendo objetos: 100% (29/29), listo.
Resolviendo deltas: 100% (13/13), listo.
warning: remoto HEAD refiere a un ref inexistente, no se puede hacer checkout.

```

El que he fet a la comanda nvim fitxer.txt ha sigut esborrar els canvis que havia fet a l'arxiu per acabar amb l'error.

Pregunta 10. Visualització

```
joelghanemgomez@debian: ~  
Los siguientes paquetes se han retenido:  
  linux-headers-amd64 linux-image-amd64  
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 2 no actualizados.  
joelghanemgomez@debian:~$ sudo apt-get install gitk  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias... Hecho  
Leyendo la información de estado... Hecho  
Se instalarán los siguientes paquetes adicionales:  
  libtk8.6 tcl tcl8.6 tk tk8.6  
Paquetes sugeridos:  
  git-doc tcl-tclreadline  
Se instalarán los siguientes paquetes NUEVOS:  
  gitk libtk8.6 tcl tcl8.6 tk tk8.6  
0 actualizados, 6 nuevos se instalarán, 0 para eliminar y 2 no actualizados.  
Se necesita descargar 2.027 kB de archivos.  
Se utilizarán 4.539 kB de espacio de disco adicional después de esta operación.  
¿Desea continuar? [S/n] s  
Des:1 http://deb.debian.org/debian bullseye/main amd64 libtk8.6 amd64 8.6.11-2 [780 kB]  
Des:2 http://deb.debian.org/debian bullseye/main amd64 tk8.6 amd64 8.6.11-2 [172,3 kB]  
Des:3 http://deb.debian.org/debian bullseye/main amd64 tcl8.6 amd64 8.6.11+dfsg-1 [124 kB]  
Des:4 http://deb.debian.org/debian bullseye/main amd64 tcl amd64 8.6.11+1 [5.780 B]  
Des:5 http://deb.debian.org/debian bullseye/main amd64 tk amd64 8.6.11+1 [5.828 B]  
Des:6 http://deb.debian.org/debian bullseye/main amd64 gitk all 1:2.30.2-1+deb11u2 [1.040 kB]  
Descargados 2.027 kB en 0s (8.664 kB/s)  
Seleccionando el paquete libtk8.6:amd64 previamente no seleccionado.  
(Leyendo la base de datos ... 273970 ficheros o directorios instalados actualmente.)  
Preparando para desempaquetar .../0-libtk8.6_8.6.11-2_amd64.deb ...  
Desempaquetando libtk8.6:amd64 (8.6.11-2) ...  
Seleccionando el paquete tk8.6 previamente no seleccionado.  
Preparando para desempaquetar .../1-tk8.6_8.6.11-2_amd64.deb ...  
Desempaquetando tk8.6 (8.6.11-2) ...  
Seleccionando el paquete tcl8.6 previamente no seleccionado.  
Preparando para desempaquetar .../2-tcl8.6_8.6.11+dfsg-1_amd64.deb ...  
Desempaquetando tcl8.6 (8.6.11+dfsg-1) ...  
Seleccionando el paquete tcl previamente no seleccionado.  
Preparando para desempaquetar .../3-tcl_8.6.11+1_amd64.deb ...  
Desempaquetando tcl (8.6.11+1) ...  
Seleccionando el paquete tk previamente no seleccionado.  
Preparando para desempaquetar .../4-tk_8.6.11+1_amd64.deb ...  
Desempaquetando tk (8.6.11+1) ...  
Seleccionando el paquete gitk previamente no seleccionado.  
Preparando para desempaquetar .../5-gitk_1:2.30.2-1+deb11u2_all.deb ...  
Desempaquetando gitk (1:2.30.2-1+deb11u2) ...  
Configurando tcl8.6 (8.6.11+dfsg-1) ...  
Configurando libtk8.6:amd64 (8.6.11-2) ...  
Configurando tcl (8.6.11+1) ...  
Configurando tk8.6 (8.6.11-2) ...  
Configurando tk (8.6.11+1) ...  
Configurando gitk (1:2.30.2-1+deb11u2) ...  
Procesando disparadores para man-db (2.9.4-2) ...  
Procesando disparadores para libc-bin (2.31-13+deb11u6) ...  
joelghanemgomez@debian:~$ cd projecte/  
joelghanemgomez@debian:~/projecte$ gitk  
joelghanemgomez@debian:~/projecte$
```

Pregunta 11. L'art de la línia de comandos

```
joelghanemgomez@debian: ~/the-art-of-command-line
joelghanemgomez@debian:~/project$ cd
joelghanemgomez@debian:~$ git clone git@github.com:JoelGhanem/the-art-of-command-line.git
Clonando en 'the-art-of-command-line'...
remote: Enumerating objects: 3577, done.
remote: Total 3577 (delta 0), reused 0 (delta 0), pack-reused 3577
Recibiendo objetos: 100% (3577/3577), 2.58 MiB | 4.19 MiB/s, listo.
Resolviendo deltas: 100% (2108/2108), listo.
joelghanemgomez@debian:~$ ls
" "      copiaScript.sh      Descargas      Escritorio
52_01    Imágenes  NetBeansProjects  partido.txt    prueba.java    the-art-of-command-line  vim
ql       introprg  nvim.appimage     Penjat        prueb.java     tmp                  yarn.lock
algo.java  cosaDePrueba.class  dobleEjecucion.sh  Document sense titol - Document sense titol.pdf  gnome-browser-conne
ctor      Libros    Orchis-theme      Plantillas    Público        Vibe35.css          gnome-shell-extensi
Canals    DAW
ons       Musica   package.json      projecte      README.md      Videos
joelghanemgomez@debian:~$ cd the-art-of-command-line/
joelghanemgomez@debian:~/the-art-of-command-line$ git branch catalan
joelghanemgomez@debian:~/the-art-of-command-line$ ls
admin      CONTRIBUTING.md  README-cs.md  README-el.md  README-fr.md  README-it.md  README-ko.md  README
E-pl.md    README-ro.md    README-sl.md  README-zh-Hant.md
AUTHORS.md  cowsay.png      README-de.md  README-es.md  README-id.md  README-ja.md  README.md      README
E-pt.md    README-ru.md    README-uk.md  README-zh.md
joelghanemgomez@debian:~/the-art-of-command-line$ nvim README-ca.mc
joelghanemgomez@debian:~/the-art-of-command-line$ git co
commit      config
joelghanemgomez@debian:~/the-art-of-command-line$ git add --all
joelghanemgomez@debian:~/the-art-of-command-line$ git commit -am "README en catalan"
[master 2641565] README en catalan
1 file changed, 244 insertions(+)
create mode 100644 README-ca.mc
joelghanemgomez@debian:~/the-art-of-command-line$ git push
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 12 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 10.03 KiB | 10.03 MiB/s, listo.
Total 3 (delta 1), reusado 0 (delta 0), pack-reusado 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:JoelGhanem/the-art-of-command-line.git
f79f79c..2641565 master -> master

joelghanemgomez@debian:~/the-art-of-command-line$ git rm README-c
README-ca.mc README-cs.md
joelghanemgomez@debian:~/the-art-of-command-line$ git rm README-ca.mc
rm 'README-ca.mc'
joelghanemgomez@debian:~/the-art-of-command-line$ ls
admin      CONTRIBUTING.md  README-cs.md  README-el.md  README-fr.md  README-it.md  README-ko.md  README
E-pl.md    README-ro.md    README-sl.md  README-zh-Hant.md
AUTHORS.md  cowsay.png      README-de.md  README-es.md  README-id.md  README-ja.md  README.md      README
E-pt.md    README-ru.md    README-uk.md  README-zh.md
joelghanemgomez@debian:~/the-art-of-command-line$ git checkout
catalan     HEAD          master        origin/HEAD   origin/master
joelghanemgomez@debian:~/the-art-of-command-line$ git checkout catalan
Cambiado a rama 'catalan'
joelghanemgomez@debian:~/the-art-of-command-line$ nvim
admin/      cowsay.png      README-cs.md  README-es.md  README-it.md  README.md
          README-ro.md    README-uk.md
AUTHORS.md  .git/           README-de.md  README-fr.md  README-ja.md  README-pl
.md         README-ru.md    README-zh-Hant.md
CONTRIBUTING.md  .gitignore     README-el.md  README-id.md  README-ko.md  README-pt
.md         README-sl.md    README-zh.md
joelghanemgomez@debian:~/the-art-of-command-line$ git add --all
joelghanemgomez@debian:~/the-art-of-command-line$ git commit -am "README-ca"
[catalan f0898ce] README-ca
1 file changed, 246 insertions(+)
create mode 100644 README-ca.md
joelghanemgomez@debian:~/the-art-of-command-line$ git push
fatal: La rama actual catalan no tiene una rama upstream.
Para realizar un push de la rama actual y configurar el remoto como upstream, use

git push --set-upstream origin catalan

joelghanemgomez@debian:~/the-art-of-command-line$ git push --set-upstream origin catalan
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 12 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 10.02 KiB | 10.02 MiB/s, listo.
Total 3 (delta 1), reusado 0 (delta 0), pack-reusado 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'catalan' on GitHub by visiting:
remote:   https://github.com/JoelGhanem/the-art-of-command-line/pull/new/catalan
remote:
To github.com:JoelGhanem/the-art-of-command-line.git
* [new branch]   catalan -> catalan
Rama 'catalan' configurada para hacer seguimiento a la rama remota 'catalan' de 'origin'.
joelghanemgomez@debian:~/the-art-of-command-line$ git push
Everything up-to-date
joelghanemgomez@debian:~/the-art-of-command-line$
```

L'URL del meu repositori:

<https://github.com/JoelGhanem/the-art-of-command-line.git>
<git@github.com:JoelGhanem/the-art-of-command-line.git>

gh repo clone JoelGhanem/the-art-of-command-line

Conclusions

El control de versions ha millorat molt la vida dels programadors ja que ara actualitzar software ja no és tan perillós com abans.

El Git és una eina molt bona, tot i que sembla que hi ha gent que la considera massa difícil si segueixes les pautes correctes va genial, el fet de que has d'anar amb compte m'agrada ja que m'obliga a pensar bé el que he de fer en comptes de llençar-me de cap a fer les coses.

Aquest exercici m'ha ajudat a documentar i aprendre sobre l'ús del Git. L'utilitzaré com a referència futura si tinc cap problema, ha estat molt extens però he après molt.