Git Assignment

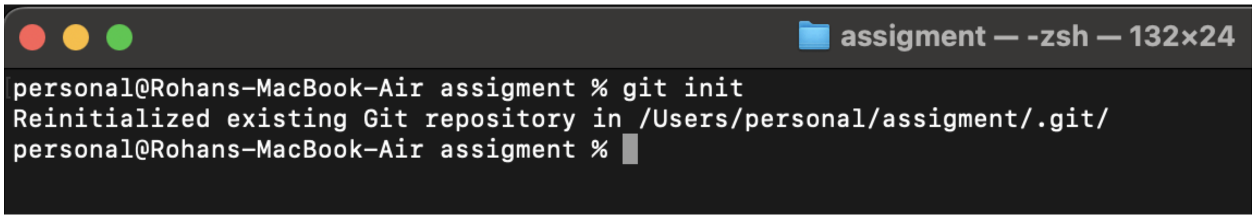
Name: Rohan Vashisht

Roll No.: 150096724132

Cohort: Jenson Huang

1) The most basic command for initializing a git repository is:

`git init`



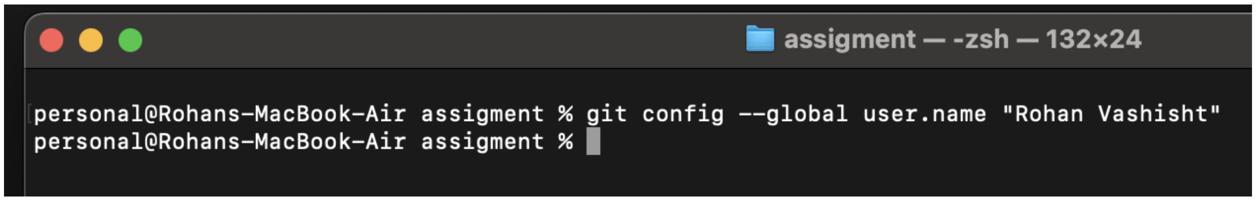
```
personal@Rohans-MacBook-Air assigment % git init
Reinitialized existing Git repository in /Users/personal/assigment/.git/
personal@Rohans-MacBook-Air assigment %
```

This command is used to initialize a new/existing repository from Git

The other initializing commands consists of configuring the basic details about the git user which consists of the following data:

To configure the basic user name we can use the following command:

`git config --global user.name "Rohan Vashisht"`

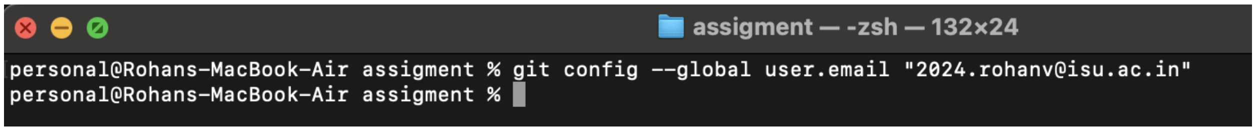


```
personal@Rohans-MacBook-Air assigment % git config --global user.name "Rohan Vashisht"
personal@Rohans-MacBook-Air assigment %
```

This will set the user for the specific user.

To configure the basic user email we can use the following command:

`git config --global user.email "2024.rohanv@isu.ac.in"`

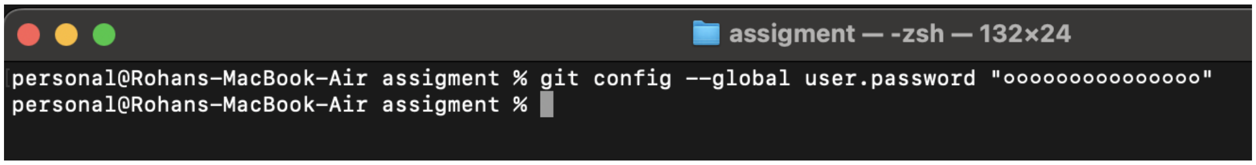


```
personal@Rohans-MacBook-Air assigment % git config --global user.email "2024.rohanv@isu.ac.in"
personal@Rohans-MacBook-Air assigment % 
```

This is used to configure the user email for future commits.

To configure the password for the user:

`git config --global user.password "∘∘∘∘∘∘∘∘∘∘∘∘∘∘∘"`

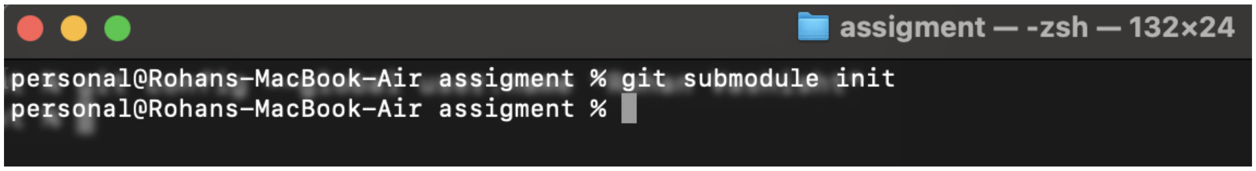


```
personal@Rohans-MacBook-Air assigment % git config --global user.password "ooooooooooooooo"
personal@Rohans-MacBook-Air assigment % 
```

This is used to add a password for the user.

Also, to configure any submodules within an existing repository, we can do the following:

`git submodule init`

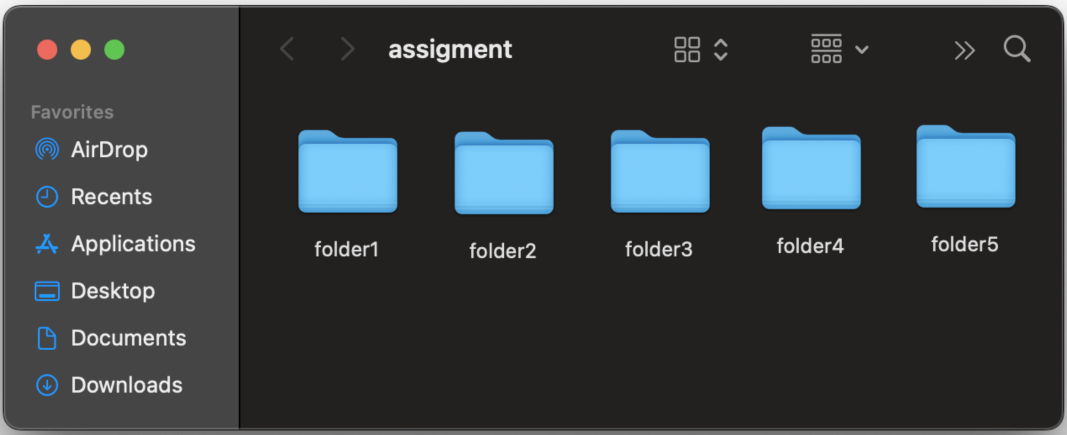


```
personal@Rohans-MacBook-Air assigment % git submodule init
personal@Rohans-MacBook-Air assigment % 
```

This initialized any submodules that were previously added to git.

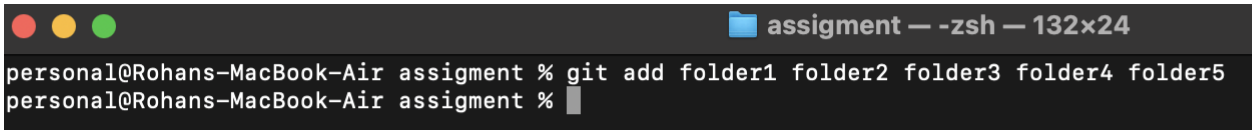
2) Create folder using git commands.

First we need to create 5 folders



After this, we can add these folders to git using:

`git add folder1 folder2 folder3 folder4 folder5`



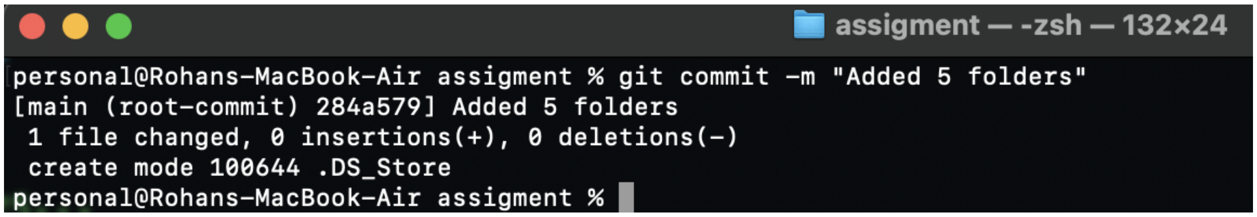
```
personal@Rohans-MacBook-Air assigment % git add folder1 folder2 folder3 folder4 folder5
personal@Rohans-MacBook-Air assigment %
```

This adds the folder to git.

Then we need to commit these changes onto github:

For that we can:

`git commit -m "Added 5 folders"`



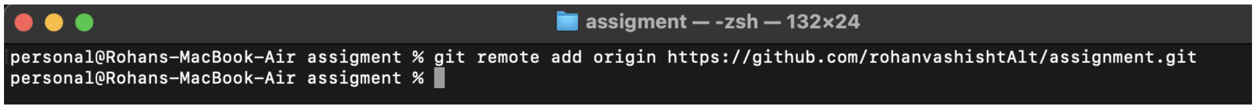
```
personal@Rohans-MacBook-Air assigment % git commit -m "Added 5 folders"
[main (root-commit) 284a579] Added 5 folders
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .DS_Store
personal@Rohans-MacBook-Air assigment %
```

Then we need to switch to main branch:

`git branch -M main`

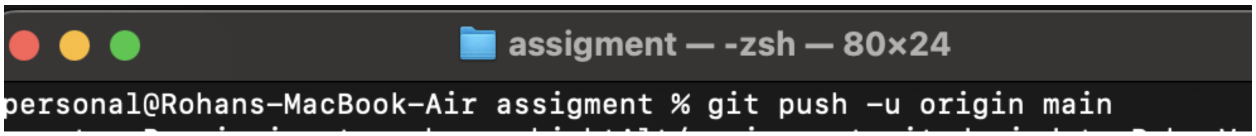
Now, Lets add the remote:

`git remote add origin` [`https://github.com/rohanvashishtAlt/assignment.git`](https://github.com/rohanvashishtAlt/assignment.git)



```
assigment — -zsh — 132×24
personal@Rohans-MacBook-Air assigment % git remote add origin https://github.com/rohanvashishtAlt/assignment.git
personal@Rohans-MacBook-Air assigment %
```

And finally git push -u origin main to push the changes onto GitHub.



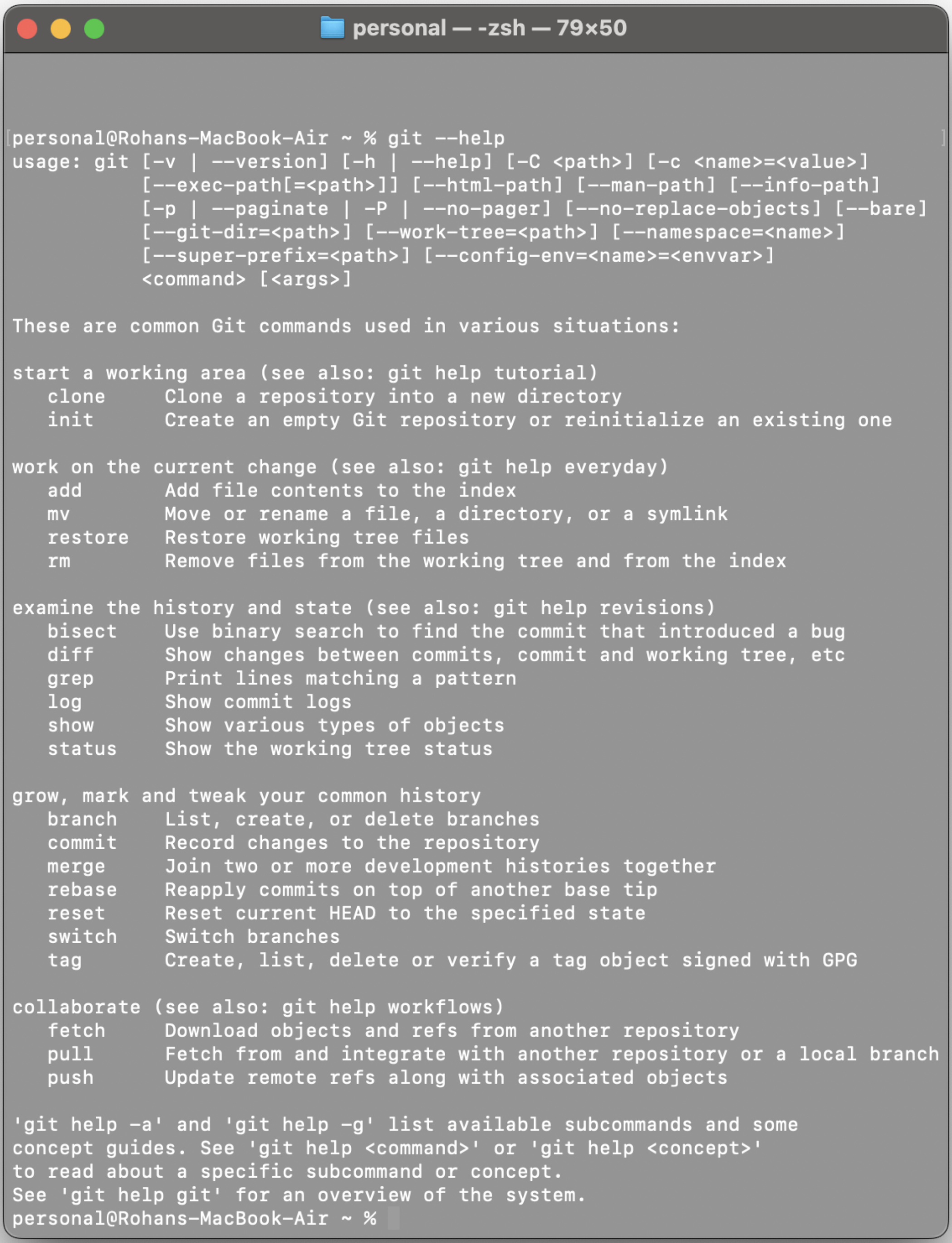
```
assigment — -zsh — 80×24
personal@Rohans-MacBook-Air assigment % git push -u origin main
```

Hence, all the changes have been uploaded onto GitHub.

Here is the link to the repository:

https://github.com/rohanvashishtAlt/assignment

3) `git --help`

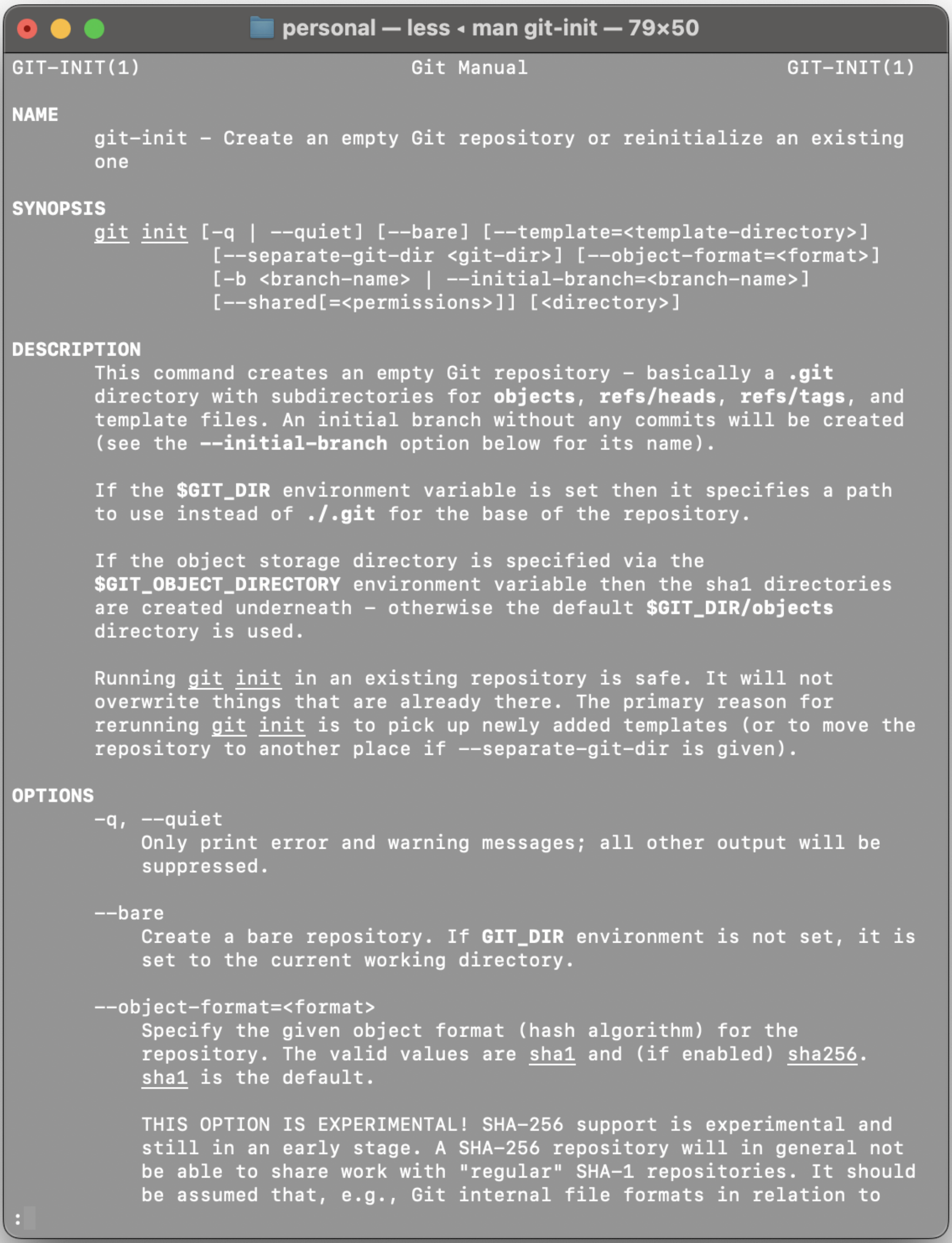


personal — -zsh — 79×50

```
personal@Rohans-MacBook-Air ~ % git --help
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone     Clone a repository into a new directory
   init      Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add       Add file contents to the index
   mv        Move or rename a file, a directory, or a symlink
   restore   Restore working tree files
   rm        Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect    Use binary search to find the commit that introduced a bug
   diff      Show changes between commits, commit and working tree, etc
   grep      Print lines matching a pattern
   log       Show commit logs
   show      Show various types of objects
   status    Show the working tree status

grow, mark and tweak your common history
   branch    List, create, or delete branches
   commit    Record changes to the repository
   merge     Join two or more development histories together
   rebase    Reapply commits on top of another base tip
   reset     Reset current HEAD to the specified state
   switch    Switch branches
   tag       Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch     Download objects and refs from another repository
   pull      Fetch from and integrate with another repository or a local branch
   push      Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
personal@Rohans-MacBook-Air ~ %
```

This command displays a general help script about git command line tool.

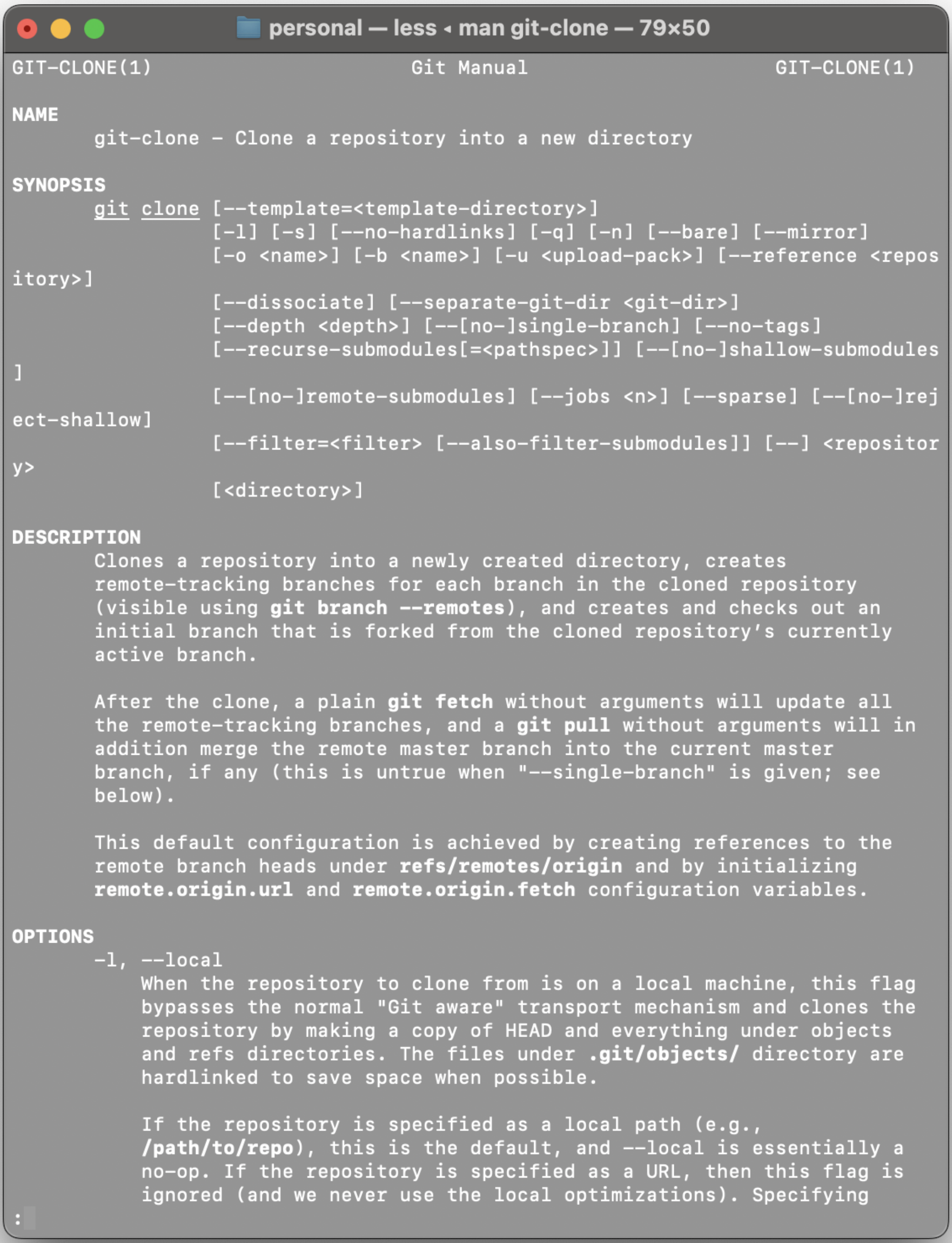
4) `git help init`



```
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 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).

OPTIONS
       -q, --quiet
           Only print error and warning messages; all other output will be
           suppressed.

       --bare
           Create a bare repository. If GIT_DIR environment is not set, it is
           set to the current working directory.

       --object-format=<format>
           Specify the given object format (hash algorithm) for the
           repository. The valid values are sha1 and (if enabled) sha256.
           sha1 is the default.

           THIS OPTION IS EXPERIMENTAL! SHA-256 support is experimental and
           still in an early stage. A SHA-256 repository will in general not
           be able to share work with "regular" SHA-1 repositories. It should
           be assumed that, e.g., Git internal file formats in relation to
:
```

This command displays a general help script about git initialization in the git cli tool.

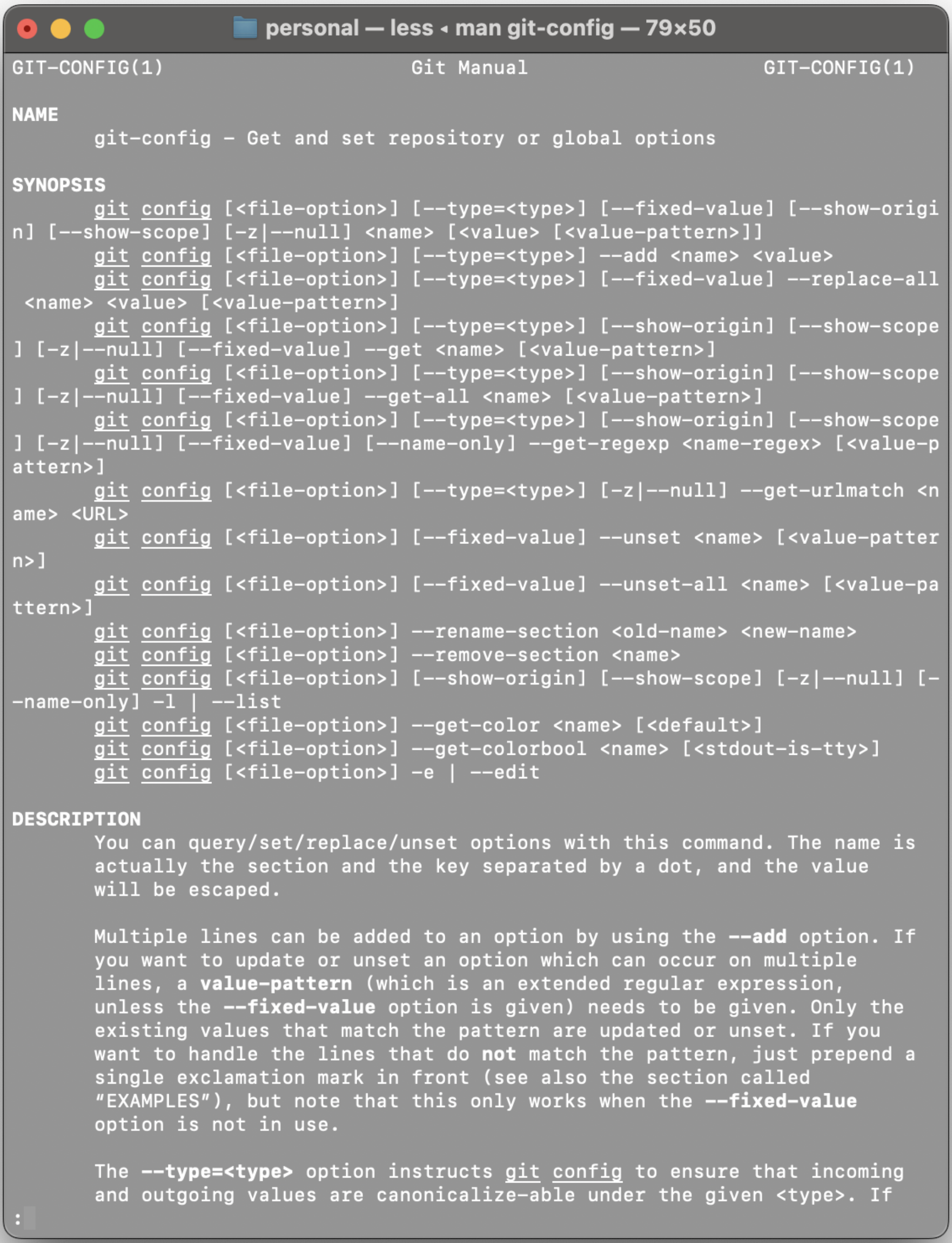
5) `git help clone`



```
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 <repos
itory>]
                 [--dissociate] [--separate-git-dir <git-dir>]
                 [--depth <depth>] [--[no-]single-branch] [--no-tags]
                 [--recurse-submodules[=<pathspec>]] [--[no-]shallow-submodules
]
                 [--[no-]remote-submodules] [--jobs <n>] [--sparse] [--[no-]rej
ect-shallow]
                 [--filter=<filter> [--also-filter-submodules]] [--] <repositor
y>
                 [<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.

OPTIONS
       -l, --local
           When the repository to clone from is on a local machine, this flag
           bypasses the normal "Git aware" transport mechanism and clones the
           repository by making a copy of HEAD and everything under objects
           and refs directories. The files under .git/objects/ directory are
           hardlinked to save space when possible.

           If the repository is specified as a local path (e.g.,
           /path/to/repo), this is the default, and --local is essentially a
           no-op. If the repository is specified as a URL, then this flag is
           ignored (and we never use the local optimizations). Specifying
:
```

Shows help script related to git's clone command.

6) `git config —help`

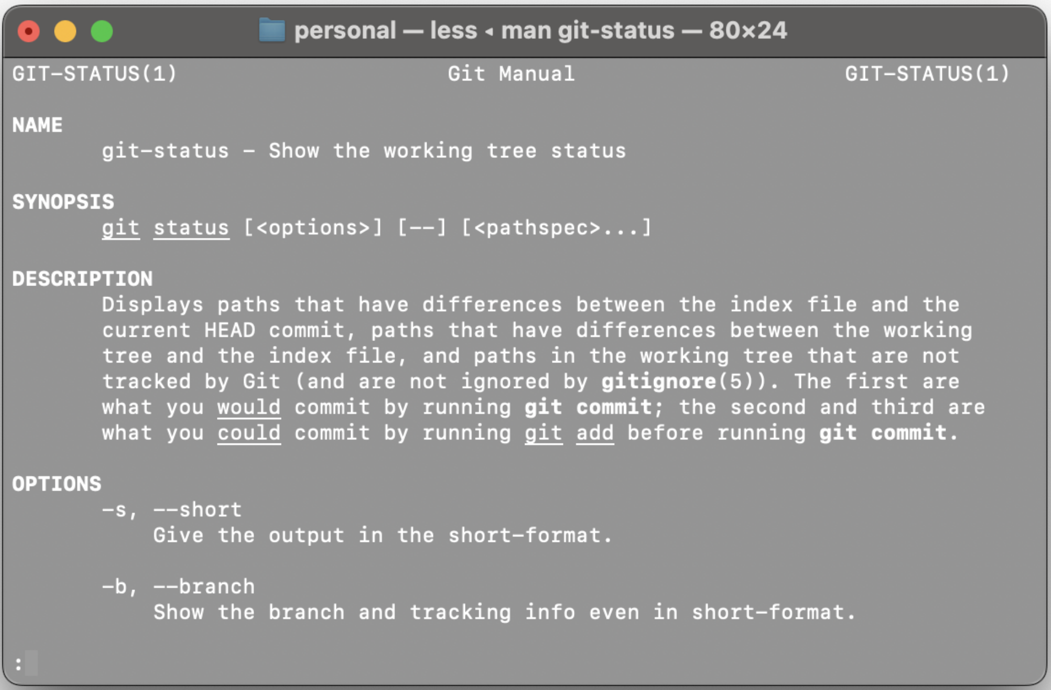


personal — less ◂ man git-config — 79×50

```
GIT-CONFIG(1)                    Git Manual                    GIT-CONFIG(1)

NAME
       git-config - Get and set repository or global options

SYNOPSIS
       git config [<file-option>] [--type=<type>] [--fixed-value] [--show-origi
n] [--show-scope] [-z|--null] <name> [<value> [<value-pattern>]]
       git config [<file-option>] [--type=<type>] --add <name> <value>
       git config [<file-option>] [--type=<type>] [--fixed-value] --replace-all
 <name> <value> [<value-pattern>]
       git config [<file-option>] [--type=<type>] [--show-origin] [--show-scope
] [-z|--null] [--fixed-value] --get <name> [<value-pattern>]
       git config [<file-option>] [--type=<type>] [--show-origin] [--show-scope
] [-z|--null] [--fixed-value] --get-all <name> [<value-pattern>]
       git config [<file-option>] [--type=<type>] [--show-origin] [--show-scope
] [-z|--null] [--fixed-value] [--name-only] --get-regexp <name-regex> [<value-p
attern>]
       git config [<file-option>] [--type=<type>] [-z|--null] --get-urlmatch <n
ame> <URL>
       git config [<file-option>] [--fixed-value] --unset <name> [<value-patter
n>]
       git config [<file-option>] [--fixed-value] --unset-all <name> [<value-pa
ttern>]
       git config [<file-option>] --rename-section <old-name> <new-name>
       git config [<file-option>] --remove-section <name>
       git config [<file-option>] [--show-origin] [--show-scope] [-z|--null] [-
-name-only] -l | --list
       git config [<file-option>] --get-color <name> [<default>]
       git config [<file-option>] --get-colorbool <name> [<stdout-is-tty>]
       git config [<file-option>] -e | --edit

DESCRIPTION
       You can query/set/replace/unset options with this command. The name is
       actually the section and the key separated by a dot, and the value
       will be escaped.

       Multiple lines can be added to an option by using the --add option. If
       you want to update or unset an option which can occur on multiple
       lines, a value-pattern (which is an extended regular expression,
       unless the --fixed-value option is given) needs to be given. Only the
       existing values that match the pattern are updated or unset. If you
       want to handle the lines that do not match the pattern, just prepend a
       single exclamation mark in front (see also the section called
       “EXAMPLES”), but note that this only works when the --fixed-value
       option is not in use.

       The --type=<type> option instructs git config to ensure that incoming
       and outgoing values are canonicalize-able under the given <type>. If
:
```

Shows configuration help about git’s various config based commands.

7) `git help status`



```
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.

OPTIONS
       -s, --short
           Give the output in the short-format.

       -b, --branch
           Show the branch and tracking info even in short-format.

:
```

This shows a help script related to git’s status.