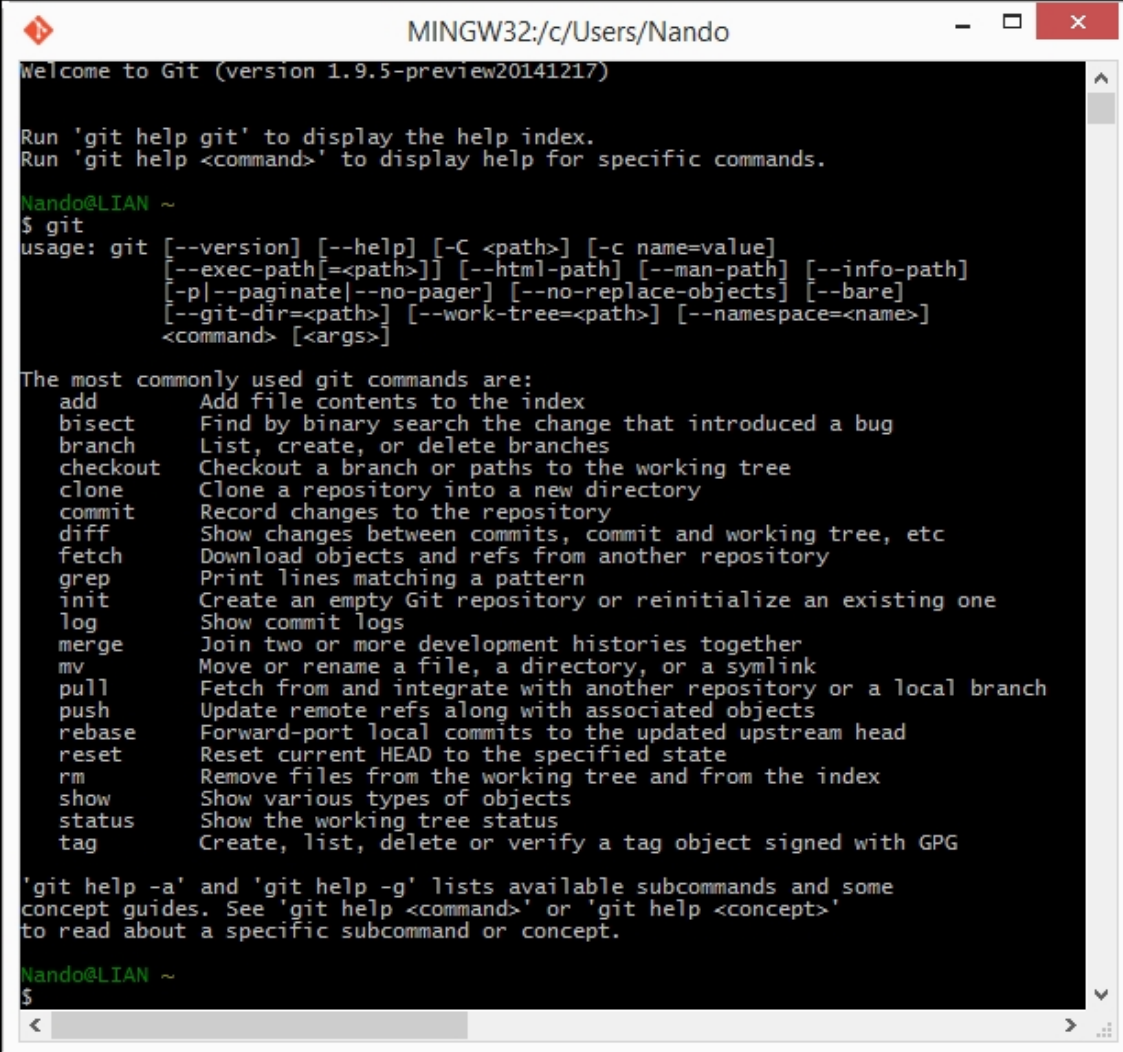


Lab: Getting Started with Collaboration

Running our first Git command

Open a prompt and simply type `git` (or the equivalent, `git --help`), as shown in the following screenshot:



```
MINGW32:/c/Users/Nando
Welcome to Git (version 1.9.5-preview20141217)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Nando@LIAN ~
$ git
usage: git [--version] [--help] [-C <path>] [-c name=value]
         [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
         [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
         [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
         <command> [<args>]

The most commonly used git commands are:
  add           Add file contents to the index
  bisect        Find by binary search the change that introduced a bug
  branch        List, create, or delete branches
  checkout       Checkout a branch or paths to the working tree
  clone          Clone a repository into a new directory
  commit        Record changes to the repository
  diff           Show changes between commits, commit and working tree, etc
  fetch          Download objects and refs from another repository
  grep           Print lines matching a pattern
  init           Create an empty Git repository or reinitialize an existing one
  log            Show commit logs
  merge          Join two or more development histories together
  mv            Move or rename a file, a directory, or a symlink
  pull           Fetch from and integrate with another repository or a local branch
  push           Update remote refs along with associated objects
  rebase         Forward-port local commits to the updated upstream head
  reset          Reset current HEAD to the specified state
  rm            Remove files from the working tree and from the index
  show           Show various types of objects
  status         Show the working tree status
  tag            Create, list, delete or verify a tag object signed with GPG

'git help -a' and 'git help -g' lists available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.

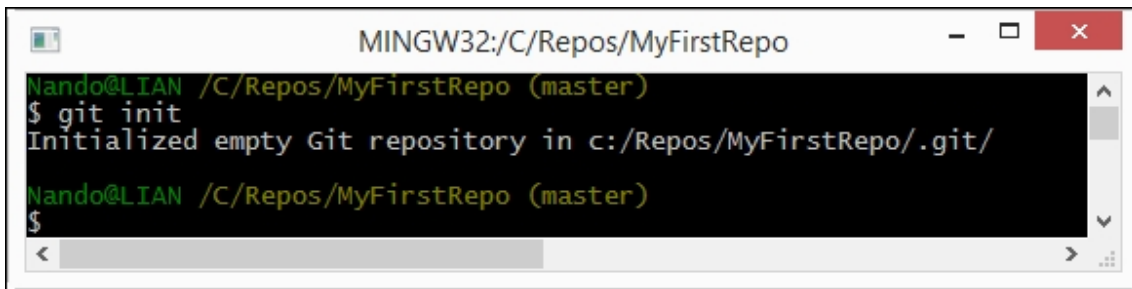
Nando@LIAN ~
$
```

So, we have Git up and running!

Setting up a new repository

The first step is to set up a new repository (or repo, for short). A **[repo]** is a container for your entire project; every file or subfolder within it belongs to that repository, in a consistent manner. Physically, a repository is nothing other than a folder that contains a special `.git` folder, the folder where the magic happens.

Let's try to make our first repo. Choose a folder you like, and type the `git init` command, as shown here:

A terminal window titled 'MINGW32:/C/Repos/MyFirstRepo' with a black background. The prompt is 'Nando@LIAN /C/Repos/MyFirstRepo (master)'. The command '\$ git init' is entered, and the output is 'Initialized empty Git repository in c:/Repos/MyFirstRepo/.git/'. The prompt returns to '\$'.

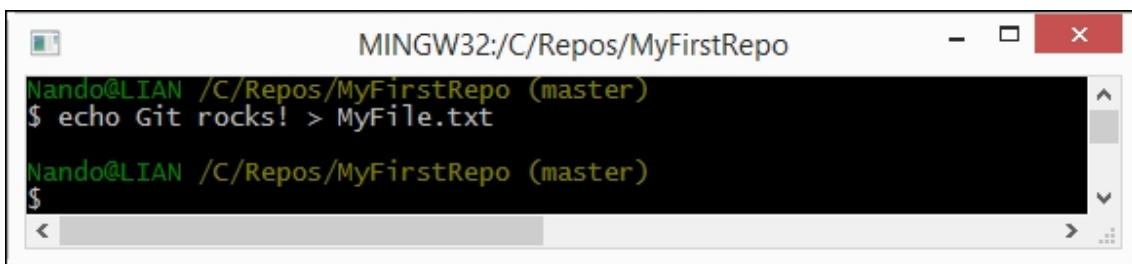
```
MINGW32:/C/Repos/MyFirstRepo
Nando@LIAN /C/Repos/MyFirstRepo (master)
$ git init
Initialized empty Git repository in c:/Repos/MyFirstRepo/.git/
Nando@LIAN /C/Repos/MyFirstRepo (master)
$
```

```
cd ~/work
mkdir MyFirstRepo
cd MyFirstRepo
git init
```

Now that we have a repo, we can start to put files inside it. Git can trace the history of any gender of files, text based or binary, small or large, with the same efficiency (more or less, large files are always a problem).

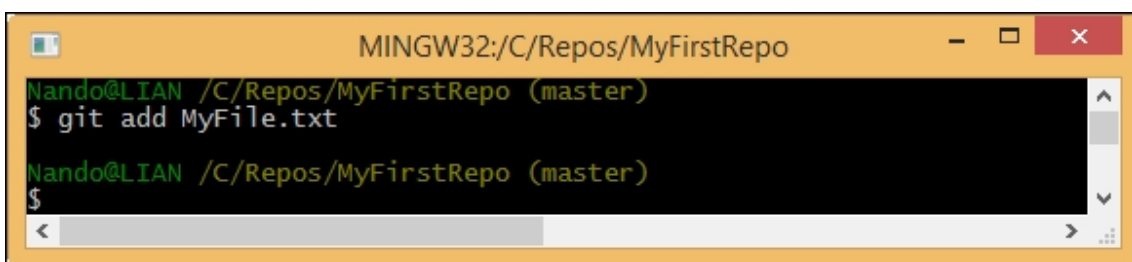
Adding a file

Let's create a text file just to give it a try.

A terminal window titled 'MINGW32:/C/Repos/MyFirstRepo' with a black background. The prompt is 'Nando@LIAN /C/Repos/MyFirstRepo (master)'. The command '\$ echo Git rocks! > MyFile.txt' is entered. The prompt returns to '\$'.

```
MINGW32:/C/Repos/MyFirstRepo
Nando@LIAN /C/Repos/MyFirstRepo (master)
$ echo Git rocks! > MyFile.txt
Nando@LIAN /C/Repos/MyFirstRepo (master)
$
```

We want `MyFile.txt` under the control of Git, so let's add it, as shown here:

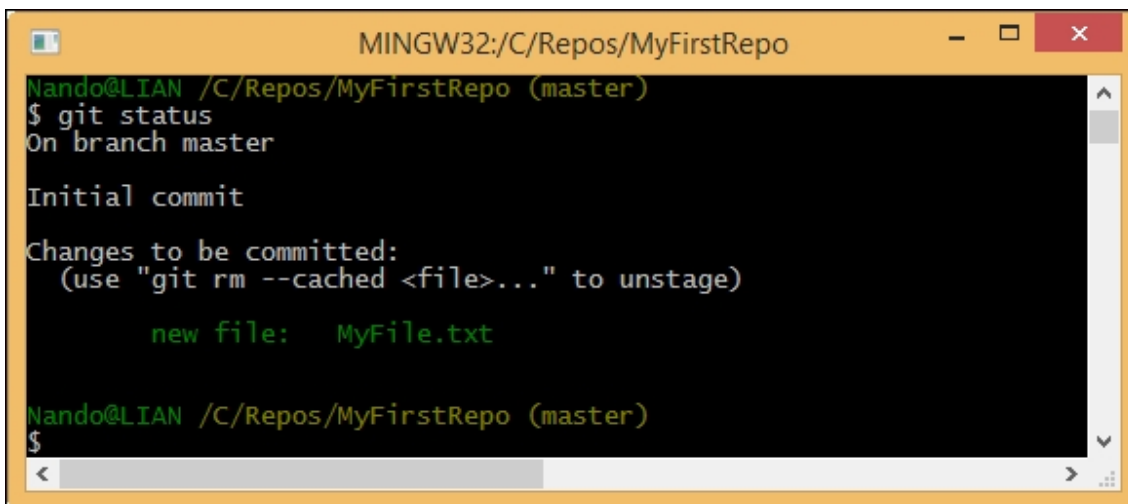
A terminal window titled 'MINGW32:/C/Repos/MyFirstRepo' with a black background. The prompt is 'Nando@LIAN /C/Repos/MyFirstRepo (master)'. The command '\$ git add MyFile.txt' is entered. The prompt returns to '\$'.

```
MINGW32:/C/Repos/MyFirstRepo
Nando@LIAN /C/Repos/MyFirstRepo (master)
$ git add MyFile.txt
Nando@LIAN /C/Repos/MyFirstRepo (master)
$
```

The `git add` command tells Git that we want it to take care of that file and check it for future modifications.

Has Git obeyed us? Let's see.

Using the `git status` command, we can check the status of the repo, as shown in the following screenshot:

A terminal window titled 'MINGW32:/C/Repos/MyFirstRepo' with a yellow title bar. The prompt is 'Nando@LIAN /C/Repos/MyFirstRepo (master)'. The command '\$ git status' has been executed, resulting in the following output: 'On branch master', 'Initial commit', 'Changes to be committed:', '(use "git rm --cached <file>..." to unstage)', and 'new file: MyFile.txt'. The prompt returns to '\$' at the bottom.

```
MINGW32:/C/Repos/MyFirstRepo
Nando@LIAN /C/Repos/MyFirstRepo (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

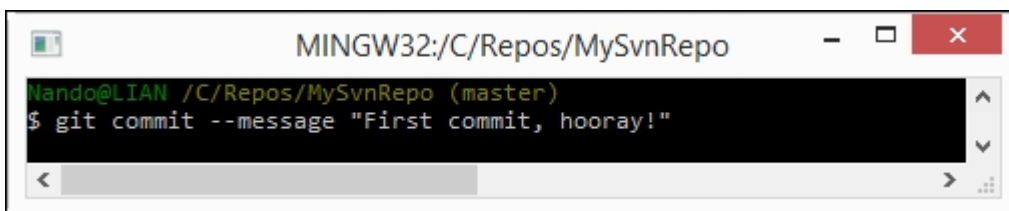
       new file:   MyFile.txt

Nando@LIAN /C/Repos/MyFirstRepo (master)
$
```

As we can see, Git has accomplished its work as expected. In this image, we can read words such as `branch` , `master` , `commit` and `unstage` . We will look at them briefly, but for the moment, let's ignore them.

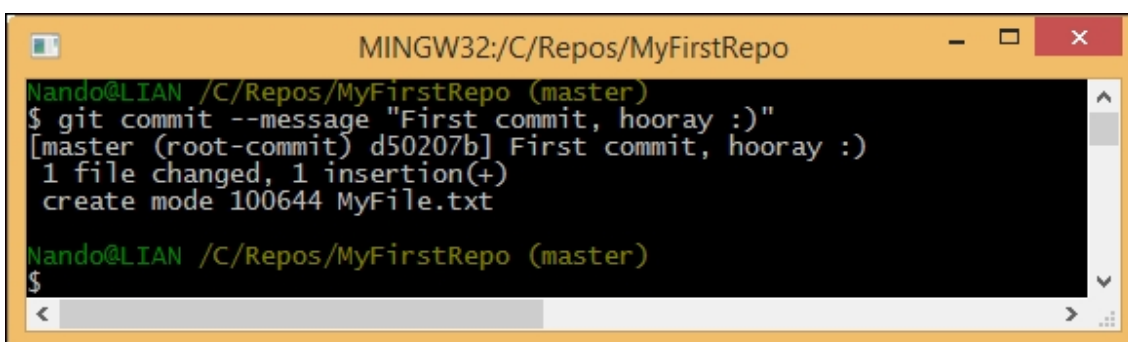
Commit the added file

At this point, Git knows about `MyFile.txt` , but we have to perform another step to fix the snapshot of its content. We have to commit it using the appropriate `git commit` command. This time, we will add some flavor to our command, using the `--message` (or `-m`) subcommand, as shown here:

A terminal window titled 'MINGW32:/C/Repos/MySvnRepo' with a white title bar. The prompt is 'Nando@LIAN /C/Repos/MySvnRepo (master)'. The command '\$ git commit --message "First commit, hooray!"' has been entered, and the cursor is at the end of the line.

```
MINGW32:/C/Repos/MySvnRepo
Nando@LIAN /C/Repos/MySvnRepo (master)
$ git commit --message "First commit, hooray!"
```

Press the `[Enter]` key.

A terminal window titled 'MINGW32:/C/Repos/MyFirstRepo' with a yellow title bar. The prompt is 'Nando@LIAN /C/Repos/MyFirstRepo (master)'. The command '\$ git commit --message "First commit, hooray :)"' has been executed, resulting in the following output: '[master (root-commit) d50207b] First commit, hooray :)', '1 file changed, 1 insertion(+)', and 'create mode 100644 MyFile.txt'. The prompt returns to '\$' at the bottom.

```
MINGW32:/C/Repos/MyFirstRepo
Nando@LIAN /C/Repos/MyFirstRepo (master)
$ git commit --message "First commit, hooray :)"
[master (root-commit) d50207b] First commit, hooray :)
1 file changed, 1 insertion(+)
create mode 100644 MyFile.txt

Nando@LIAN /C/Repos/MyFirstRepo (master)
$
```

With the commit of `MyFile.txt` , we have finally fired up our repo. It now has a `master` branch with a file within it.