**3. Key Concepts**

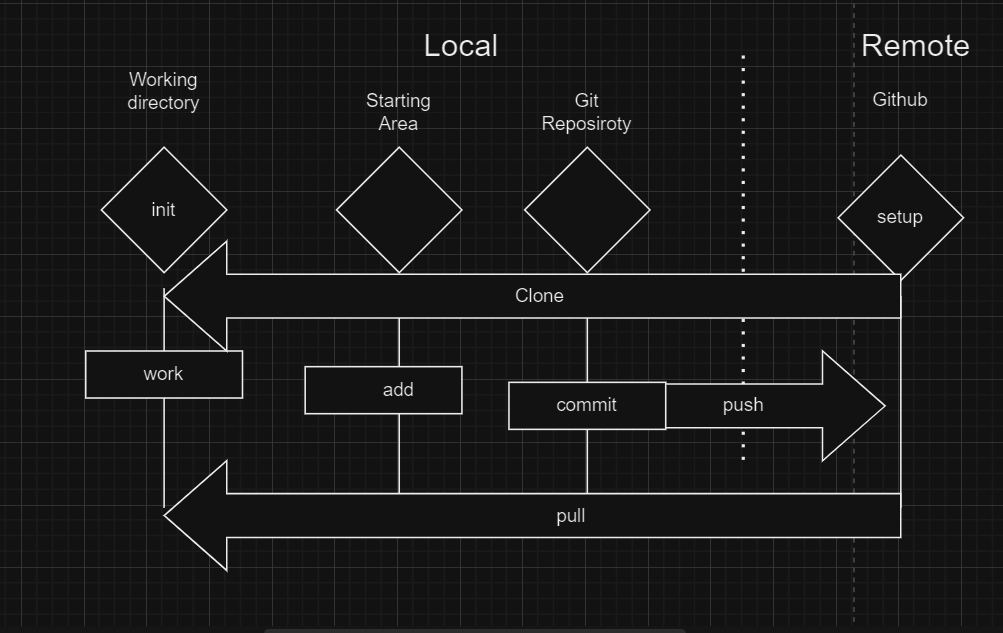
\*Repository contains files, history, config managed by Git.

\*Commit:   
Saved changes to Git repository.   
Impacts history.  
Sha1 has for unique identifier.  
  
\*Branches:  
Timeline with commits  
Master is the main, default branch  
  
\*Head:  
a pointer to last commit on branch.

\*Remote:   
Related repository but not local  
Github, Gitlab, Bitbucket and others

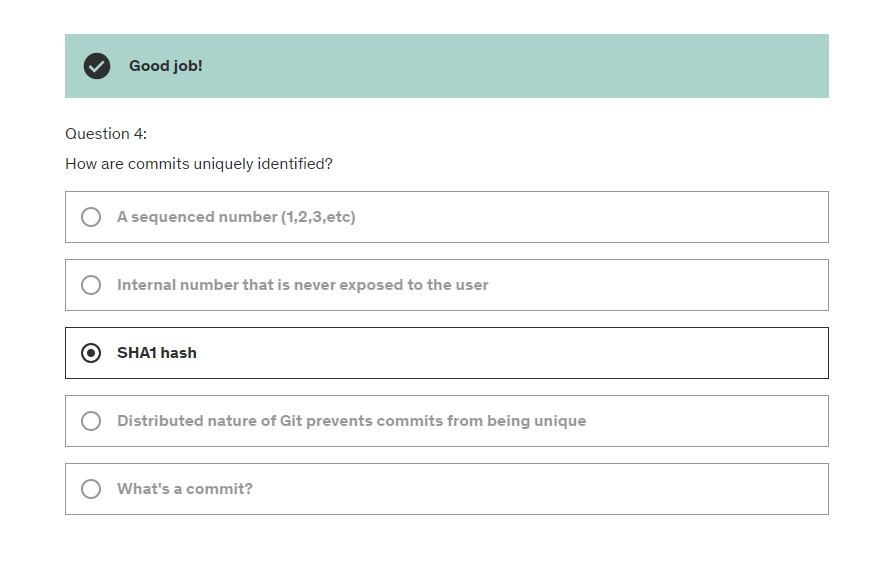
**4. Workflow**

Git Workflow starting local



There are two ways to start a Git workflow; one is local, and one is starting remote.

For starting local, we´re going o initialize our current working directory; and on the github site, you can go ahead and set up tour github repository.   
We´re going to do some type of work here; so, either editing files or creating new files in our working directory.   
Then we´re going to stage our changes using the “git add” command.  
After that, we´re going to commit our changes that are in the staging area, using commit.   
Once we´re ready to collaborate with others, we´ll “push” our changes up to our remote repository on github.

To see the benefits of everyone else´s contributions, we´ll do a “pull”, to pull in all our changes back to our local Git repository.  
  
-  
  
For starting remote, we go out to github and set up our github repository, and then do a clone to create a new repository on our local system.  
Then we would do our work, add to stagin area, and commit to our Git repository, just like before.  
When we´re ready, we´ll do our push, and then pull back.  
We´ll push our changes back up to github and pull down any changes that may have occurred.   
  
  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**8. Gitting Help – Some commands**

juanangn@JUANANGN01 MINGW64 ~

$ 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] [--no-lazy-fetch]

[--no-optional-locks] [--no-advice] [--bare] [--git-dir=<path>]

[--work-tree=<path>] [--namespace=<name>] [--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.  
  
  
git help config =  
  
[git-config(1)](file:///C:\Program%20Files\Git\mingw64\share\doc\git-doc\git-config.html)

**Help Commands**

Git Help Commands

Help commands here.

Lecture Command Listing

1. git help
2. git help config

Command Reference

Getting general help:

1. git help

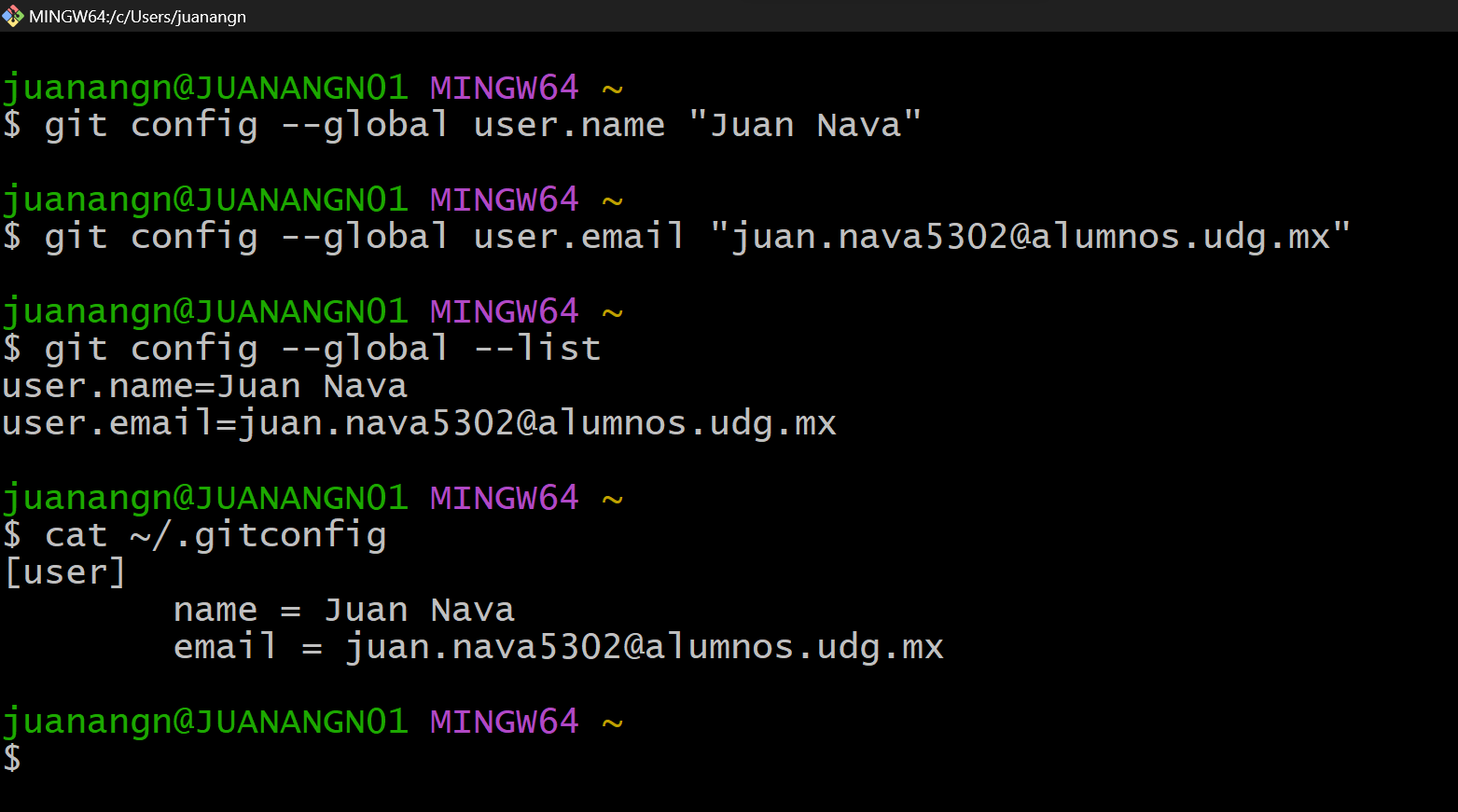
Help Command Syntax -- getting help about a specific Git command:

1. git help command

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**10 . Git Configuration**

Pwd = present working directory



**Configuration Commands**

Git Configuration Commands

What's the current directory (present working directory)?

pwd

Git Config (Global/User-level) Syntax

git config --global setting value

Configure User and Email

General Syntax:

git config --global user.name "Your Name"

git config --global user.email "you@someplace.com"

Example using course author's information:

git config --global user.name "Jason Taylor"

git config --global user.email "jason@jasongtaylor.com"

Listing All Global Configuration Settings

git config --global --list

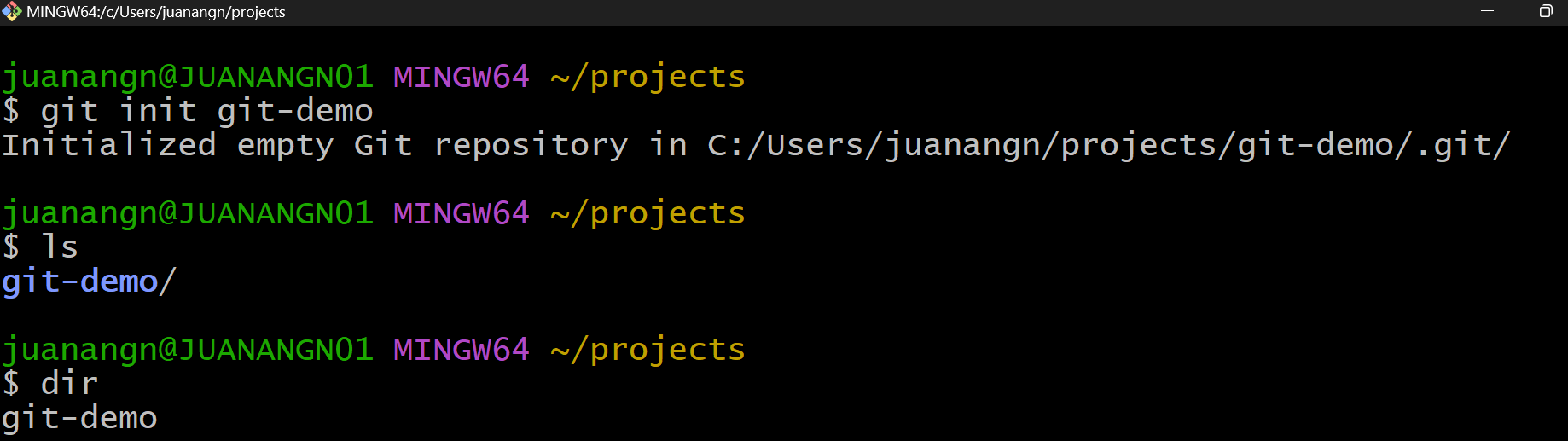
Seeing Git's User-based Config file

cat ~/.gitconfig

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

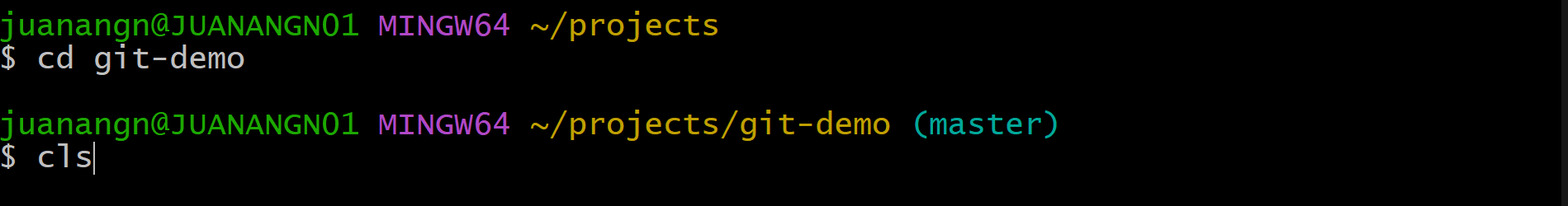
**12. Starting Fresh / From Scratch**

1: select a folder

2: git init \*project name\*  
  


Ls = dir

3: change dir into project



**13. Starting with an Existing Project**  
  
[Initializr - Start an HTML5 Boilerplate project in 15 seconds! (v1v2.io)](https://initializr.v1v2.io/)

**Starting Commands**

Git Starting Commands

Lecture Command Listing - Fresh Start

pwd

cd projects/

git init git-demo

Lecture Command Listing - Start with Existing Project

pwd

cd projects/

cd website/

ls

git init

Command Reference

Present Workding Directory

pwd

Change Directory

cd folder-name

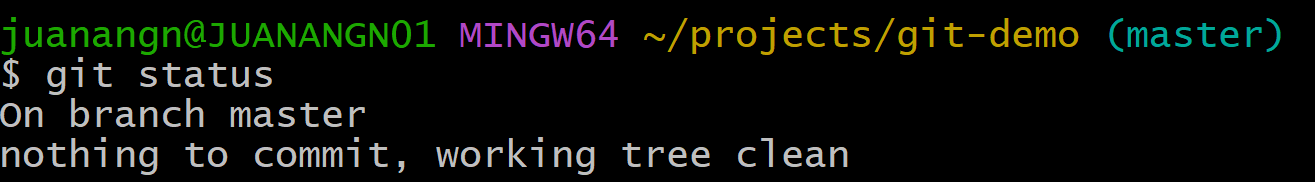
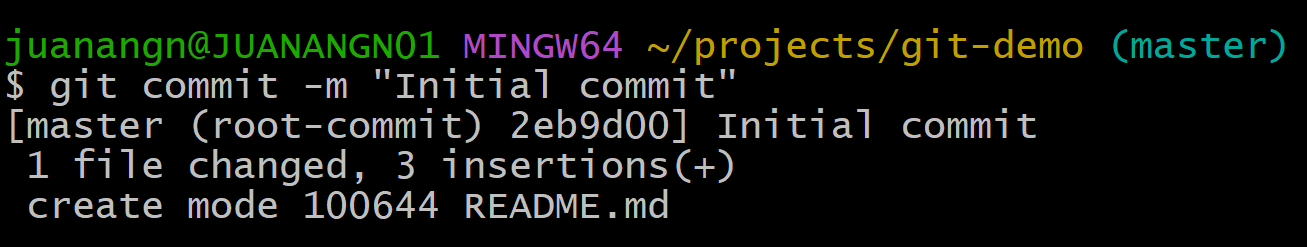
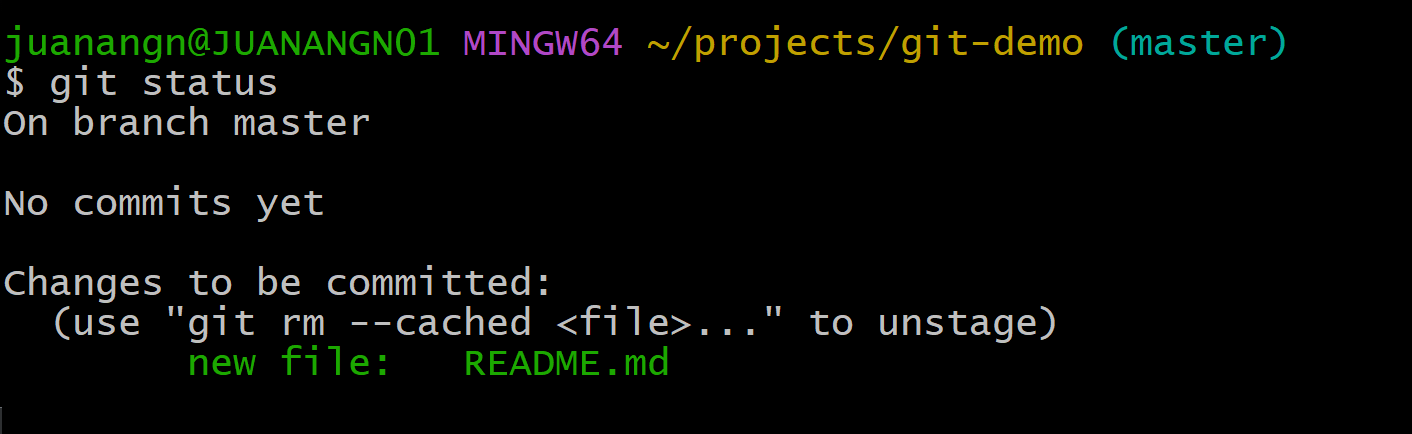
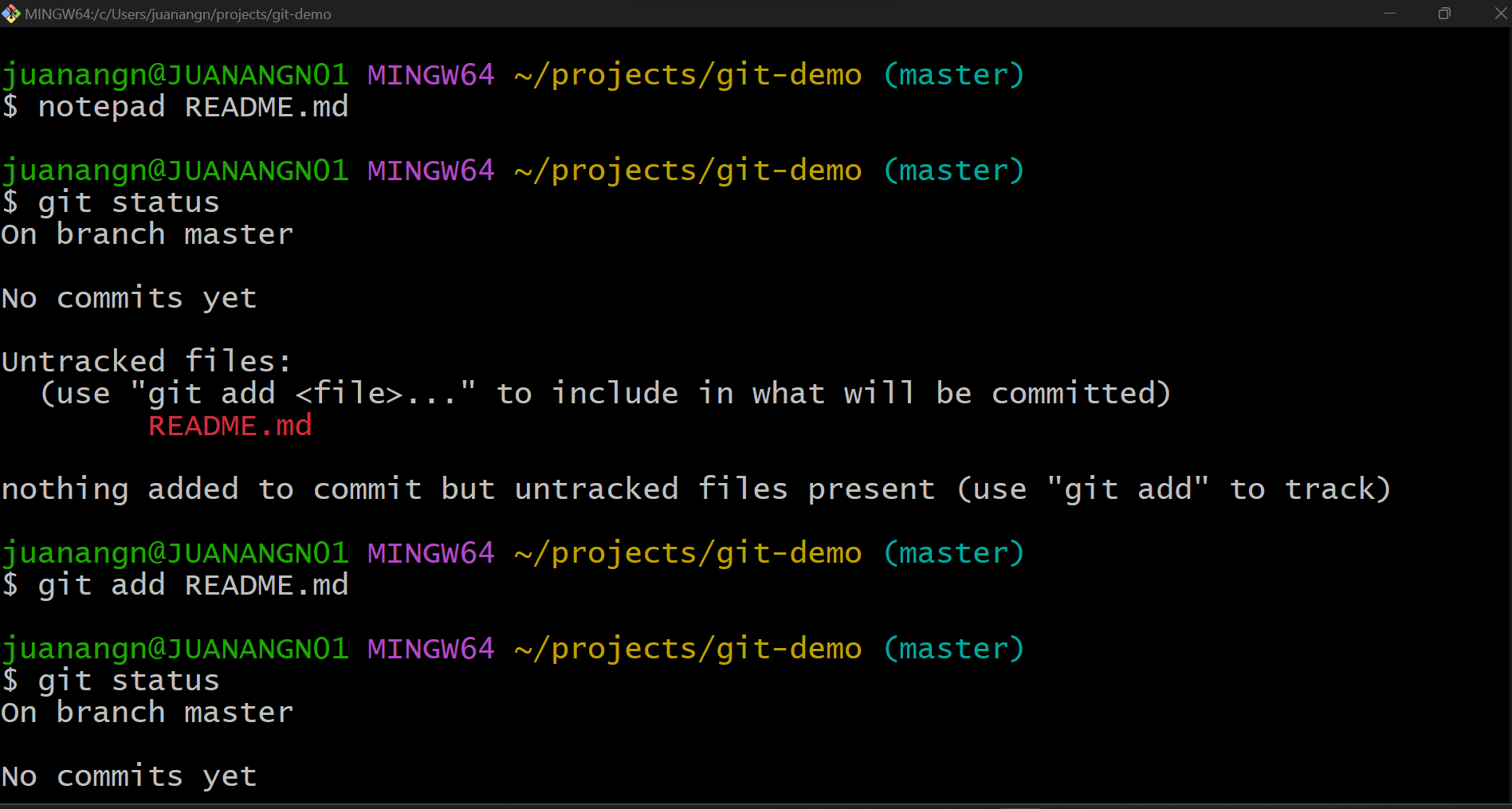
Git initialization

git init [project-name]

*project-name* parameter is optional. If not supplied, Git will initialize the current directory.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**15. First Commit**



**First Commit Commands**

#### Git First Commit Commands

##### **Lecture Command Listing**

pwd

ls

mate README.md

ls

git status

git add README.md

git status

git commit -m "Initial commit"

clear

git status

##### **Command Reference**

List

ls

Lists files and folders in current directory. Without parameters, will list non-hidden folders and files.

Git Status

git status

Shows which files have been modified in the working directory vs Git's staging area.

Git Add

git add file-name

Adds the new or newly modified file-name to Git's staging area (index).

Git Commit

git commit -m "A really good commit message"

Commits all files currently in Git's staging area. The -m parameter allows for a commit message directly from the command line.

Clear!

clear

Clears all previous commands from the terminal screen -- just a bit of clean up.

Text Mate

mate file-name  
  
code file-name

All command line demos are preformed on the MacOS. Creating and editing files is done with TextMate 2 (free) using the **mate** command from Terminal. Passing a file-name to the **mate** command will create or open that file. Windows users can use the **notepad file-name** command instead.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
17 - 18. Working locally 1,2.  
  
Once que have our project created, we can create something on it  
  
git add . = for adding multiple files

Git Working Locally Commands

Lecture Command Listing - Working Locally, Part One

pwd

git status

mate README.md

git status

git add README.md

git status

git commit -m "Adding some ipsum"

clear

git status

mate README.md

git status

git commit -am "Adding more ipsum"

git status

Lecture Command Listing - Working Locally, Part Two

pwd

git status

clear

mate index.html

git status

git add index.html

git status

mate README.md

git status

clear

git status

git add README.md

git status

git commit -m "A few changes for the website"

clear

mate README.md

mate index.html

git status

git add .

git status

git commit -m "A few more changes for website"

clear

mate README.md

git status

git add README.md

git status

git reset HEAD README.md

clear

git status

mate README.md

git checkout -- README.md

mate README.md

git status

Command Reference

Express Commit for Tracked files

git commit -am "Awesome commit message"

Use the *-a* parameter with the **git commit** command to directly commit newly modified tracked files. Warning: Only do this for small changes. Tracked files are files that have been previously added to Git (committed or staged).

Adding All Changed Files

git add .

The period parameter for the git add command will recursively add all new and newly modified files.

Unstage File

git reset HEAD file-name

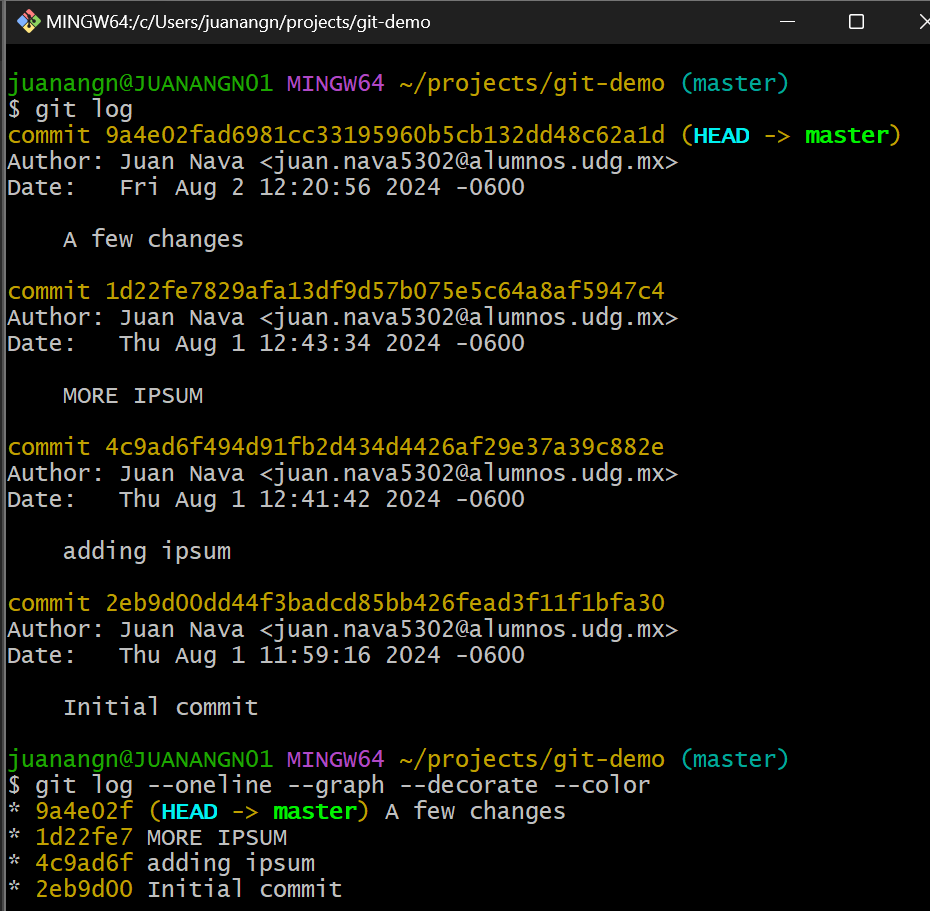
Following the above command will "unstage" the specified file from Git's staging area (aka index).

Backout Working Directory Changes

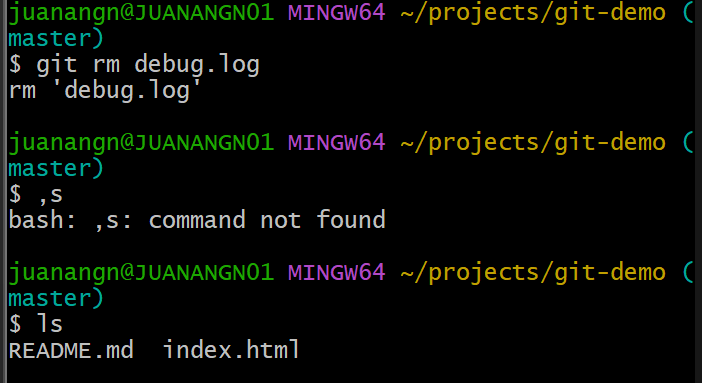
git checkout -- file-name

Following the above command will back out any changes made to the specified file and replace it with the version last committed in Git  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

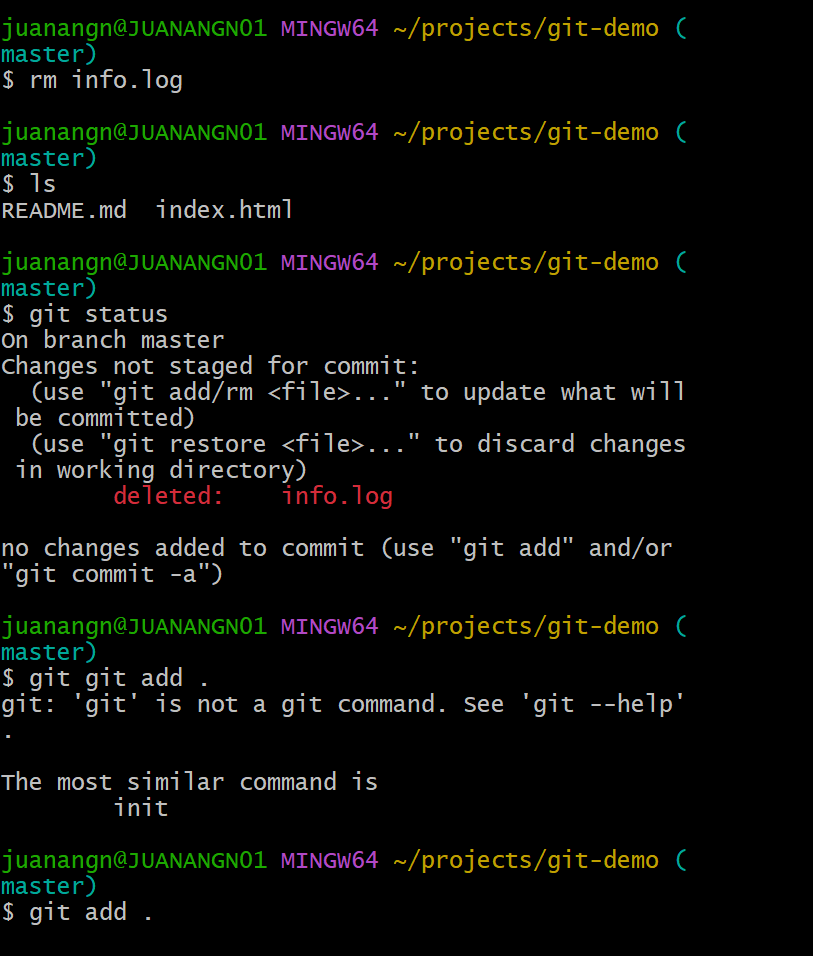
**19. historial**

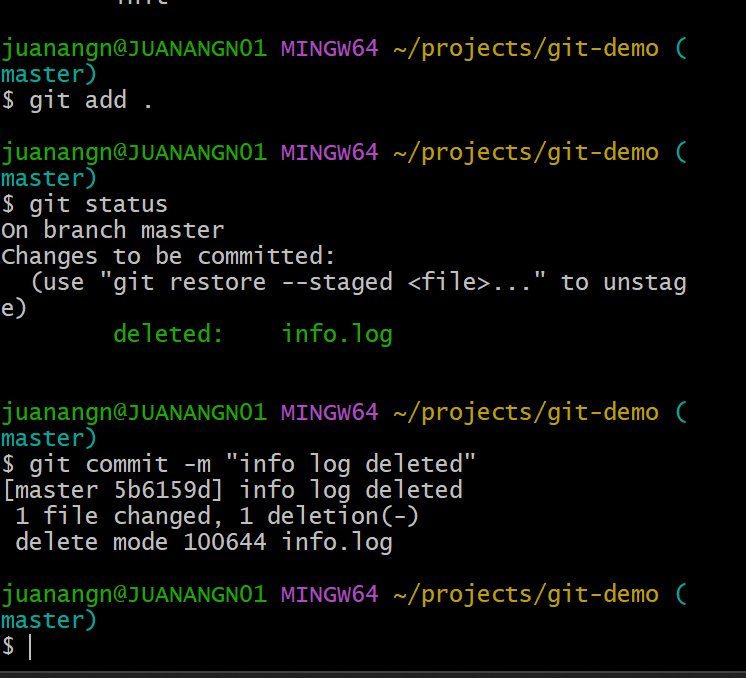
  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

21. Removing Files



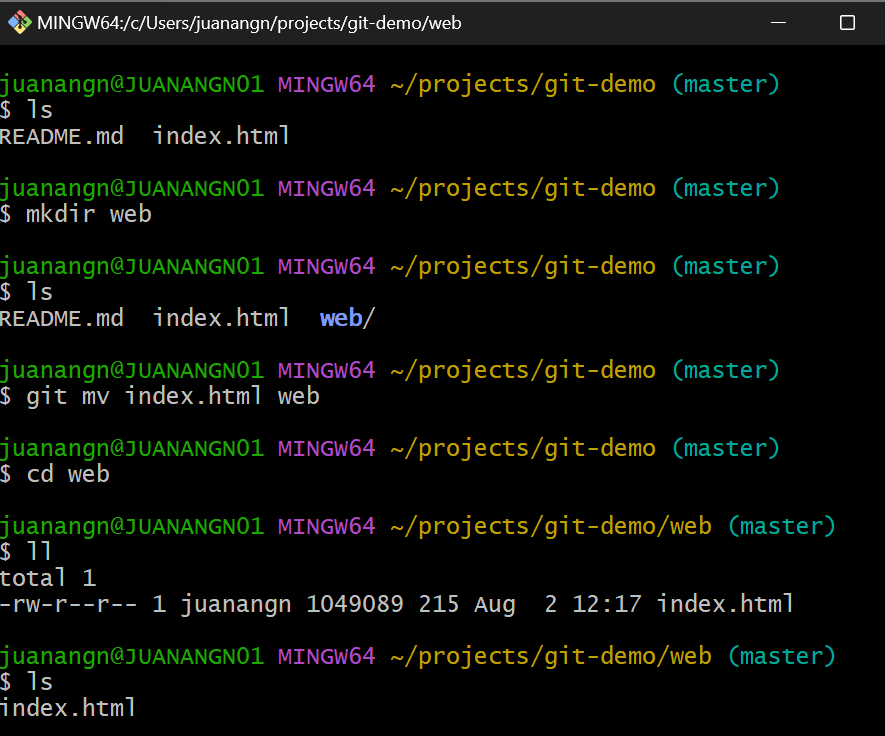
Git rm debug.log

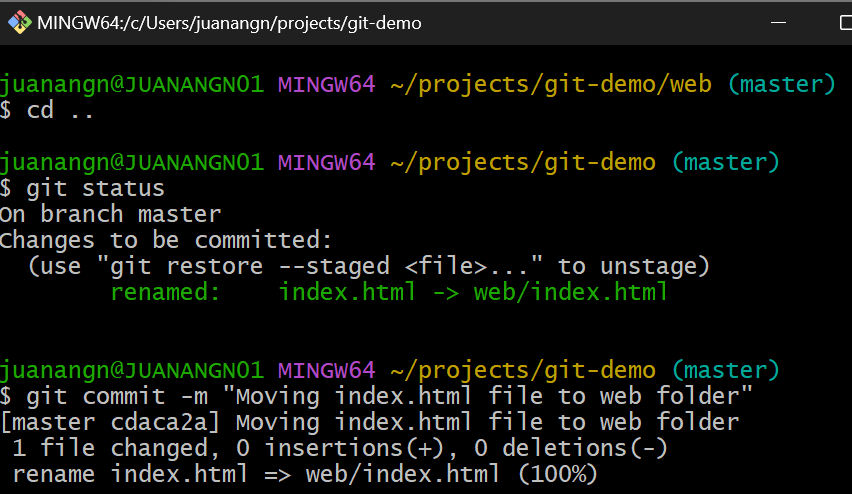




22. Moving Files

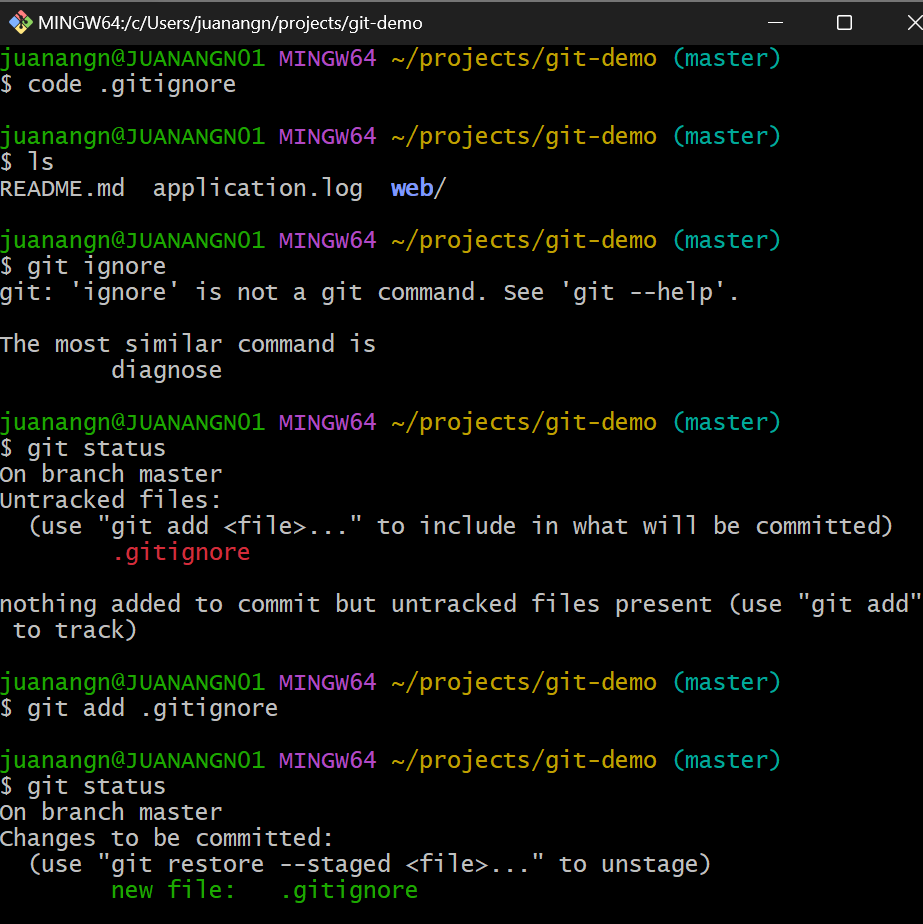
With git

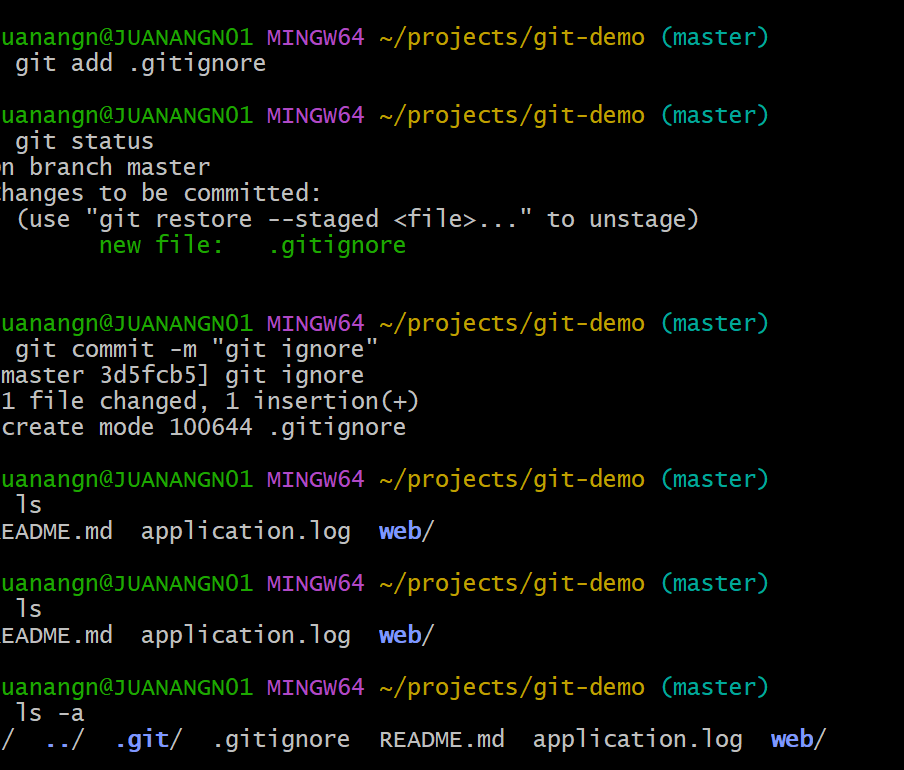




Without git

Similar to 22.  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

23. Ignoring Files   


  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

24. History and File Management Commands  
  
Git History / File Management Commands

Lecture Command Listing -- History

git log

git help log

git log --oneline --graph --decorate --color

Lecture Command Listing -- Removing Files

pwd

git status

mate debug.log

ls

git status

git add .

git status

git commit -m "adding log file that really does not belong here"

clear

git status

git rm debug.log

ls

git status

git commit -m "removing log file"

clear

mate info.log

ls

git add info.log

git commit -m "adding info log"

git status

clear

ls

rm info.log

ls

git status

git add .

git add -u

clear

git status

git commit -m "Removing info.log"

Lecture Command Listing -- Moving Files

ls

mkdir web

ls

git mv index.html web

cd web/

ll

pwd

cd ..

ls

git status

git commit -m "Moving index.html file to web folder"

clear

Lecture Command Listing -- Ignoring Files

mate application.log

ls

git status

mate .iitignore

git status

ls -a

git add .gitignore

clear

git status

git commit -m "adding ignore file"

Command Reference

Seeing Repository History

git log

git log --oneline --graph --decorate --color

Git's **log** command displays the repository's history in reverse chronological order. The no-params version displays the standard view.

Git log options from above: --oneline Compacts log data on to one line, abbreviating the SHA1 hash --graph Adds asterisk marks and pipes next to each commit to show the branching graph lines --decorate Adds the markers for branch names and tags next to corresponding commits --color Adds some color to the output -- nice to have, depending on the operating system

Removing a file using Git

git rm file-name

Removing a file using Terminal

rm file-name

This removes the file outside Git's knowledge

Updating Git's Index (staging area)

git add -u

The *-u* parameter will recursively update Git's staging area regarding deleted/moved files outside of Git.

Making a directory (folder)

mkdir folder-name

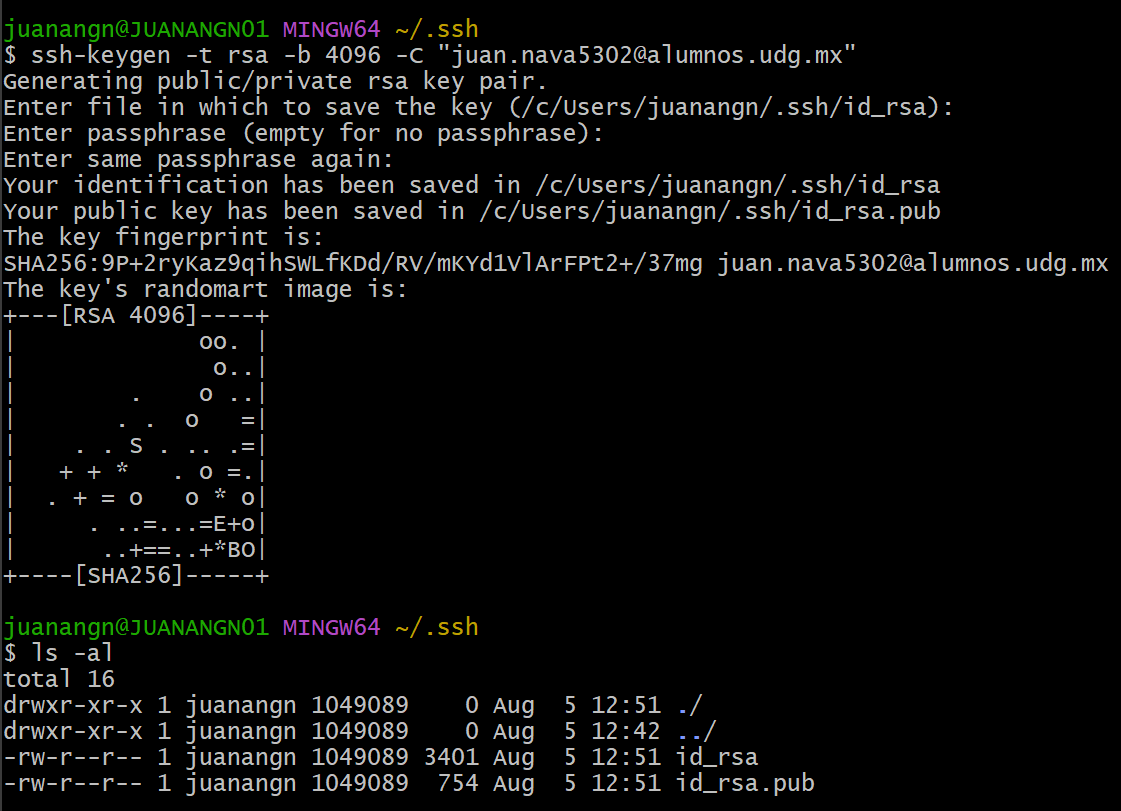
The **mkdir** command is a nearly universal command for creating a directory/folder.

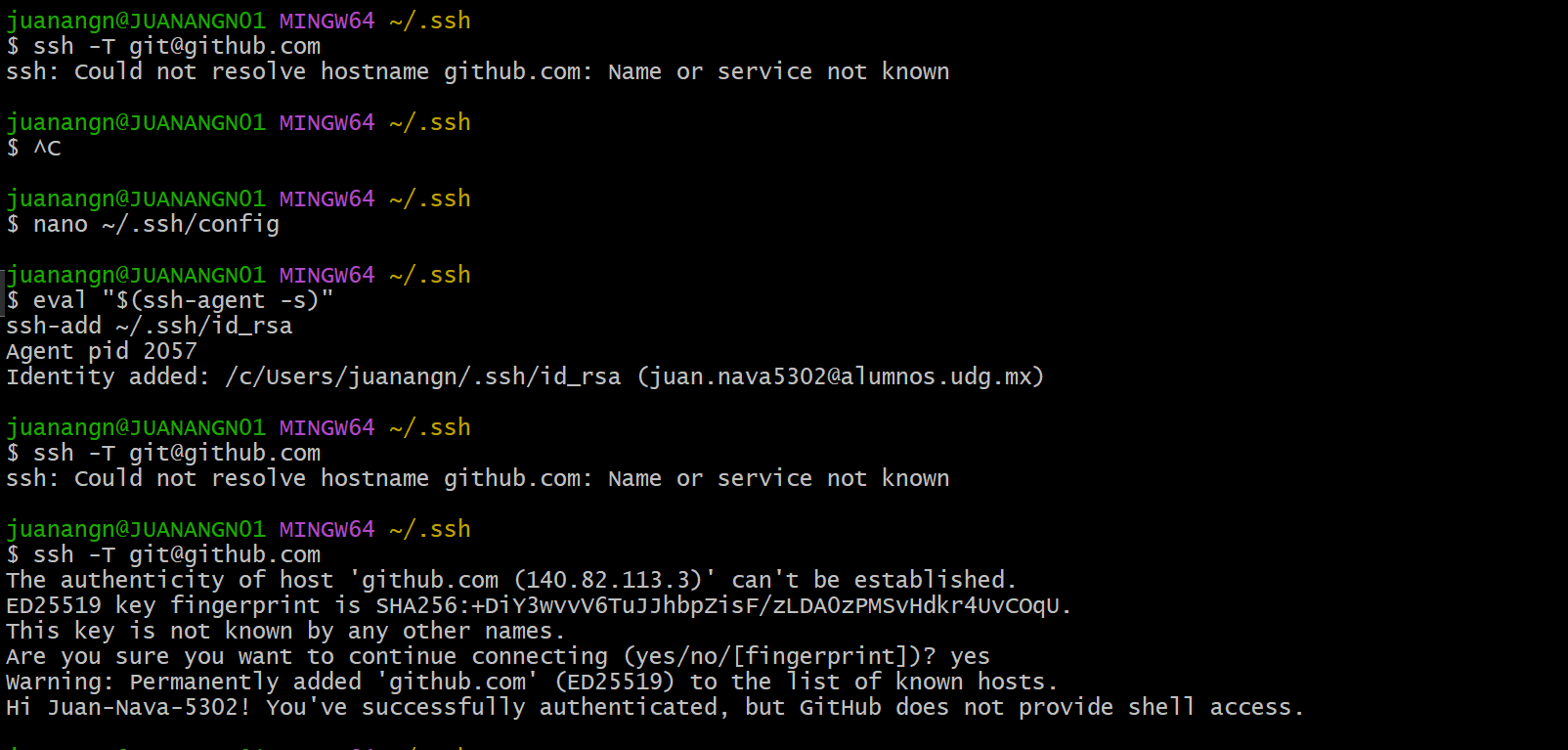
Making a directory (folder)

git mv source destination

The **git mv** command will move the *source* (file or folder) to the *destination* with Git.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
SECTION 4 GOING REMOTE

25. Setting up SSH Authentication  
  


26. SSH Authentication Commands  
  


##### **Lecture Command Listing**

cd ~

cd .ssh

mkdir .ssh

cd .ssh

pwd

ssh-keygen -t rsa -C "jason@jasongtaylor.com"

mate id\_rsa.pub

ssh -T git@github.com

##### **Command Reference**

Generating an SSH Key

ssh-keygen -t rsa -C "your.name@your-company.com"

Use your actual email address in the example above.

Verify SSH authentication

ssh -T git@github.com

Above command uses **ssh** to connect to GitHub over the SSH protocol.

27. Update! Github Changes Default Branch Name

\*Git

Default still master

Transition to something else

Options fo other default branch names

\*Github

Default now main

Can be reset back to master

More immediate transition

Option 1: Reset default master in Github

\*Github

Set default branch back to master

Setting –> repositories –> repository default branch

Change to master and Update

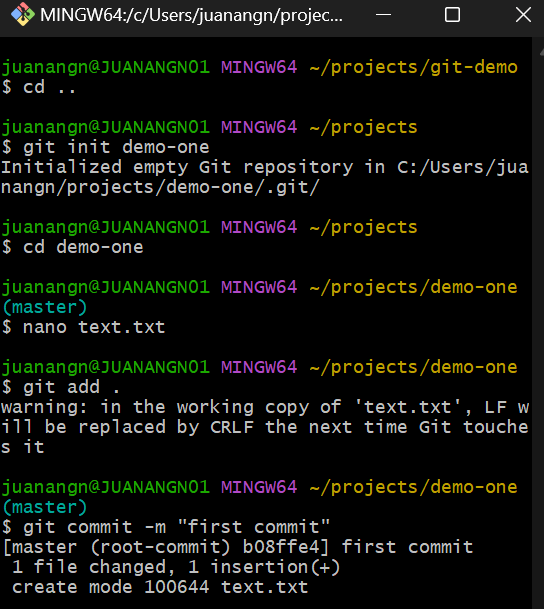
After course, change back to main (if desired)

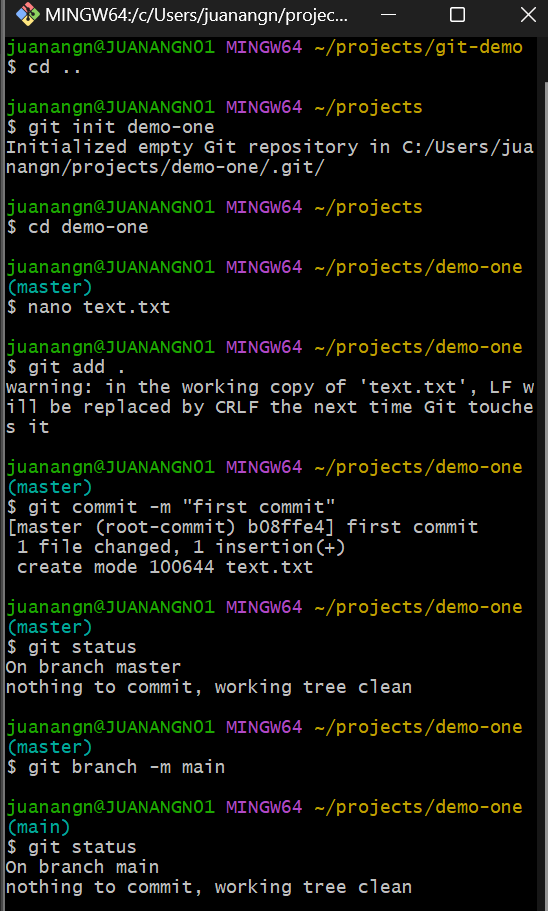
We go to github, go to settings, replace main for master

For course motives, we will only use MASTER branch

Option 2: Embrace the change and use main in Git

We need to do this in the folder called “projects”/any folder in which we keep our projects stored.

In the example, we make a new folder for the project, we locate into it and we init a new project, we need at least one file.   
  




EXAMPLE WITH MAIN IN BOTH

[Repositories (github.com)](https://github.com/settings/repositories)

https://github.com/settings/repositories

28. Collaborating With Others – Git Remotes and Github

1-Create a repository on github  
  
“git-demo-remote”

2-También tenemos que tener algo en el local.

3-Seleccionamos ssh

4-“or pus han existing repository form the command line”

Usar commando 1 y commando 3 (tomando en cuenta que las branches están sincronizadas).

IMPORTANTE, NO TENER NINGÚN CAMBIO PENDIENTE (ADD, COMMIT)

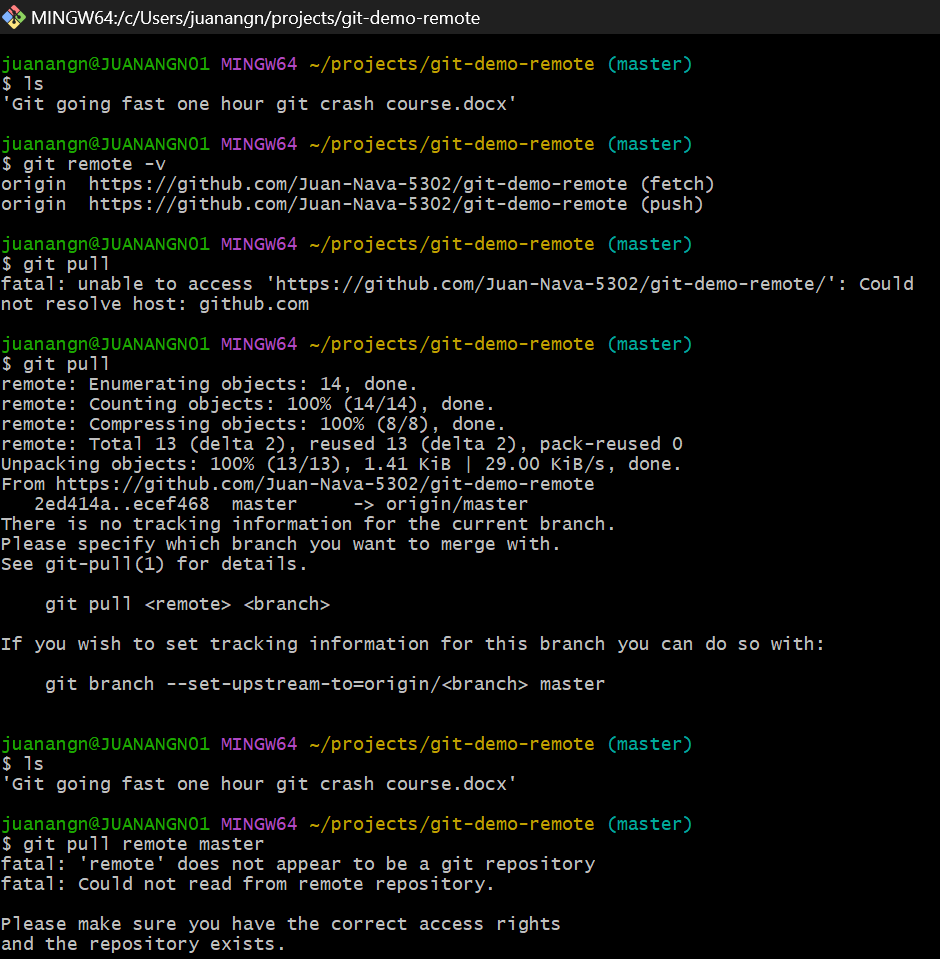
5- por cada cambio, un add, commit, pull (origin master) y push (origin master).

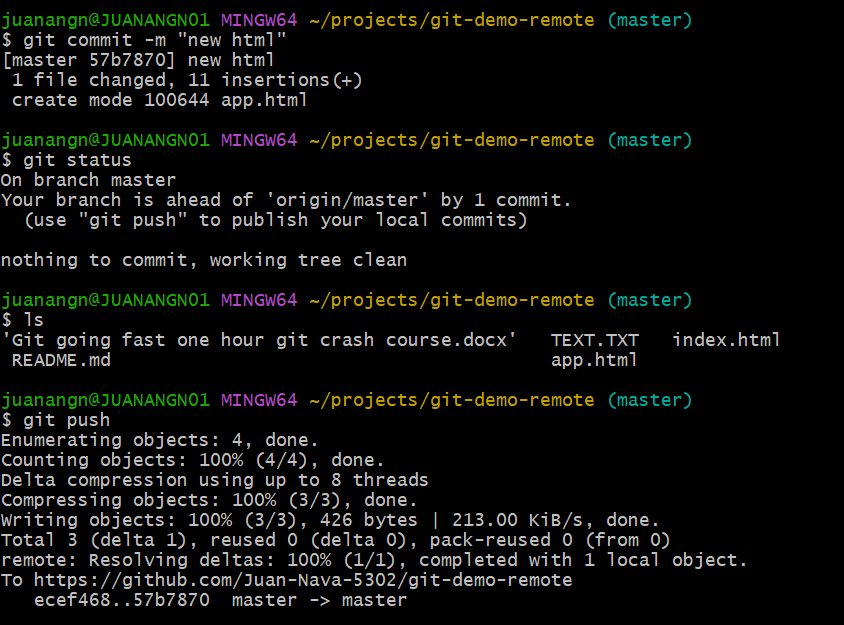
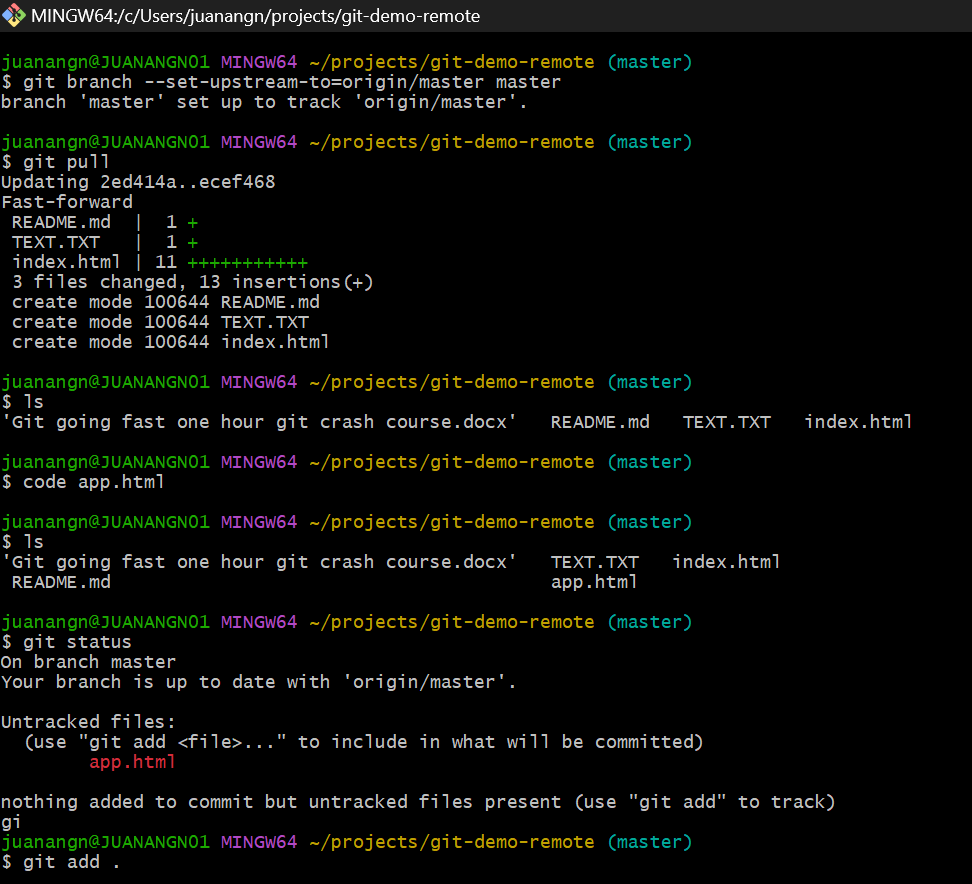
Git Remote Commands

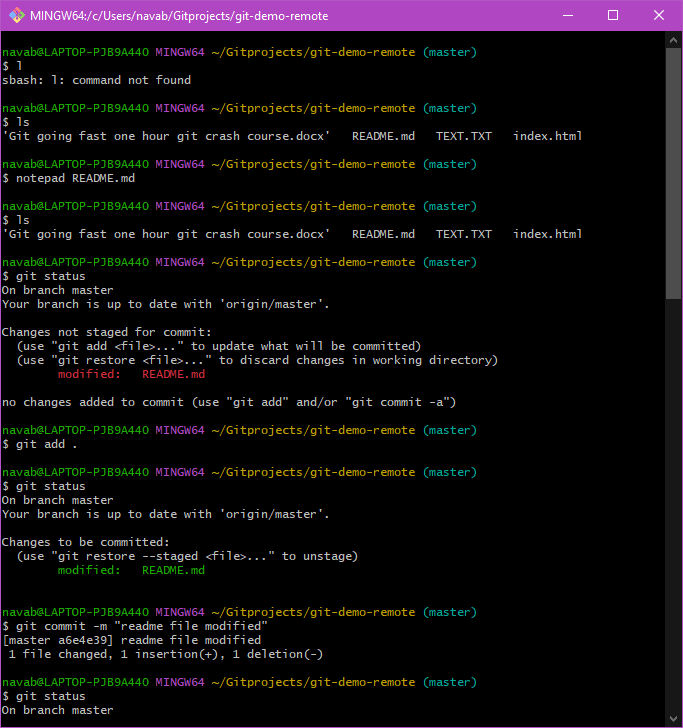
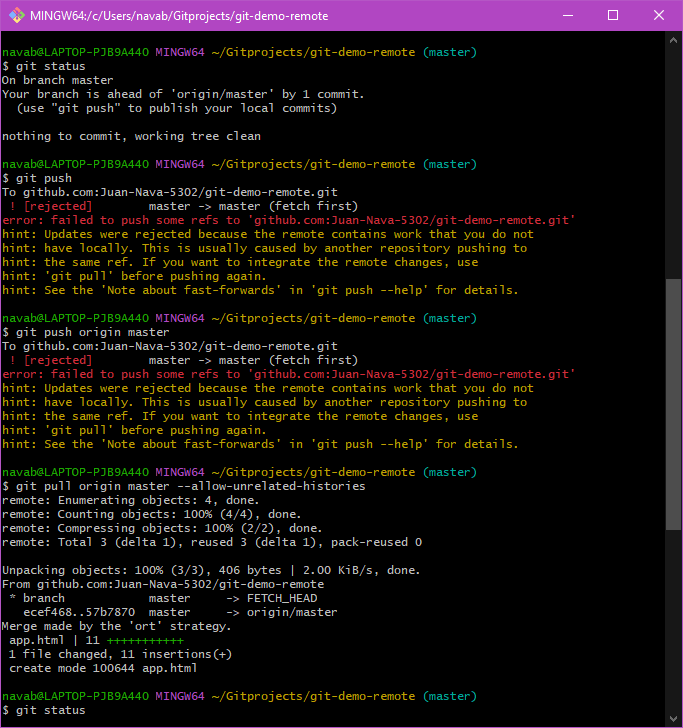
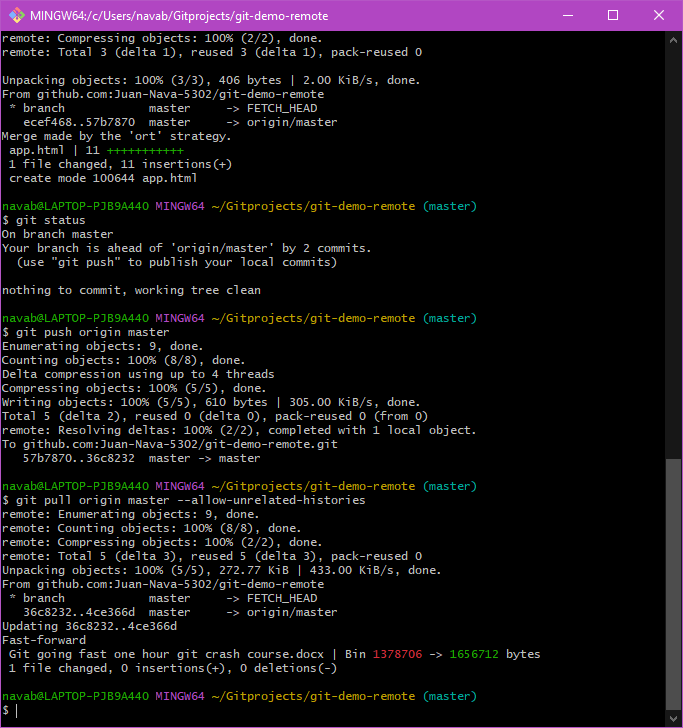
FREEE BOOK   
[Git - Book (git-scm.com)](https://git-scm.com/book/en/v2)

POSIBLES COMANDOS EXTRA:

\*DESDE LAPTOP AM





\*DESDE LAPTOP PERSONAL  
  
    
  
En resumen: los 2 comandos que problablemente requiramos sean:   
  
git pull origin master –allow-unrelated-histories

Git push origin master