

Git Introduction & Installation

Welcome to the New Collaboration!



This work by M Alexander Jurkat is licensed under a
[Creative Commons Attribution 3.0 Unported License](https://creativecommons.org/licenses/by/3.0/).

Git What?

- If you have not done so already, review the descriptions, materials, and video detailed in the Introduction to Git portion of section 1.8 Git and GitHub on Blackboard.
- You will gain an understanding of Git, version control, and why it's important.

Pro Git Book

- For this exercise, we will be working through the first two sections of the Pro Git Book, written by Scott Chacon and published by Apress.
- Browse to the table of contents for Pro Git at <http://git-scm.com/book>.
- We will be working in the Getting Started and Git Basics chapters.

More Background

- Review [Chapter 1 Getting Started](#)
- Review [Section 1.1 About Version Control](#)
- Review [Section 1.2 A Short History of Git](#)
- Review [Section 1.3 Git Basics](#)
- You can navigate these sections using the “Prev” and “Next” links at the bottom of the page.

Installing Git

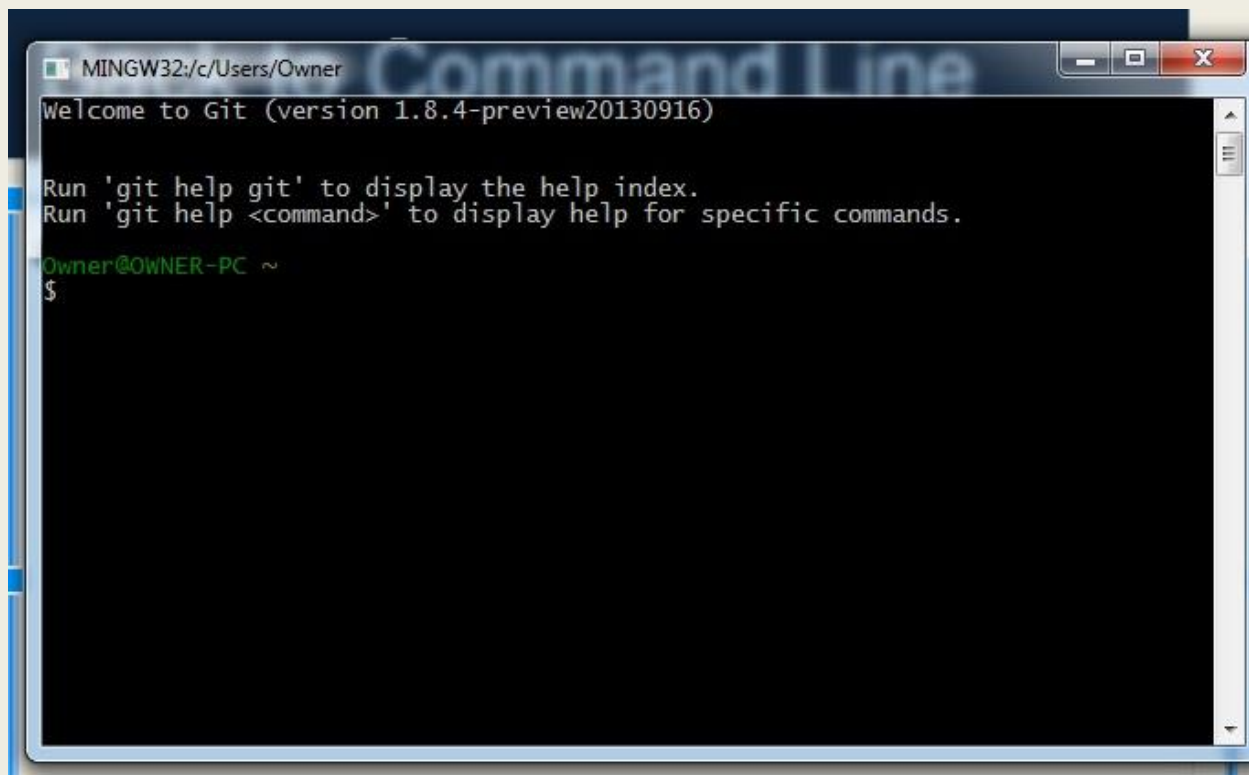
- The first step is getting Git installed on your local machine. Review [Section 1.4 Installing Git](#).
- If you feel comfortable compiling software from source, follow the instructions in the first portion of Section 1.4.
- If you don't know what that means, follow the instructions in the subsequent sections for your OS (Linux, Mac, or Windows).
- When using a Mac, try downloading from <http://git-scm.com/download/mac>.
- When working through the install, simply accept the default choices.

Back to Command Line

- Once installed, you need to start up the Git agent. It's called Git Bash.
- In the Start menu or Finder of your local machine, you should find Git Bash. Open that agent.
- The result should look familiar if you've completed section 1.7. You are at a command line prompt. Rather than being on the class shared server, however, you are on your local machine.

Back to Command Line

- Once Git Bash is open, it should look something like this:

A screenshot of a Git Bash terminal window. The window title bar shows 'MINGW32:/c/Users/Owner'. The terminal content displays a welcome message for Git version 1.8.4-preview20130916, followed by instructions on how to use 'git help'. The prompt 'Owner@OWNER-PC ~' is shown in green, followed by a '\$' symbol on the next line.

```
MINGW32:/c/Users/Owner
Welcome to Git (version 1.8.4-preview20130916)

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

Owner@OWNER-PC ~
$
```

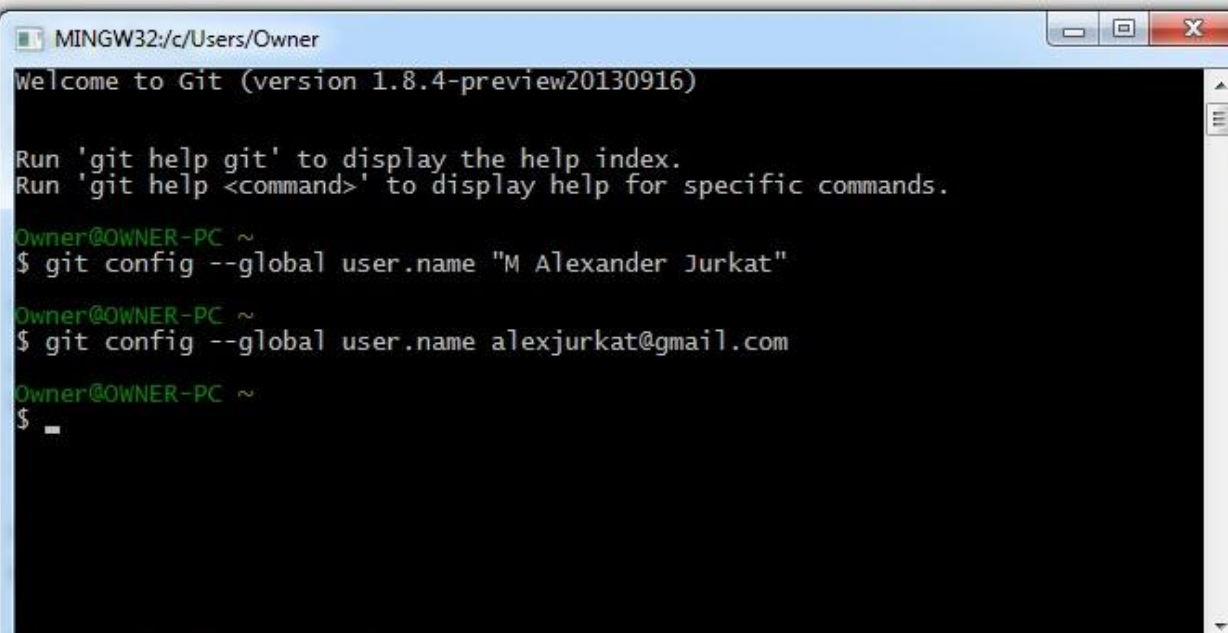
Who Are You?

Your Identity

The first thing you should do when you install Git is to set your user name and e-mail address. This is important because every Git commit uses this information, and it's immutably baked into the commits you pass around:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Again, you need to do this only once if you pass the `--global` option, because then Git will always use that information for anything you do on that system. If you want to override this with a different name or e-mail address for specific projects, you can run the command without the `--global` option when you're in that project.

A screenshot of a terminal window titled 'MINGW32:/c/Users/Owner'. The window shows the output of 'git config' commands. It starts with a welcome message for Git version 1.8.4-preview20130916, followed by instructions on how to use 'git help'. Then, two 'git config --global' commands are entered and executed: one for setting the user name to 'M Alexander Jurkat' and another for setting the user email to 'alexjurkat@gmail.com'. The prompt returns to '\$' after each command.

```
MINGW32:/c/Users/Owner
Welcome to Git (version 1.8.4-preview20130916)

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

Owner@OWNER-PC ~
$ git config --global user.name "M Alexander Jurkat"

Owner@OWNER-PC ~
$ git config --global user.email alexjurkat@gmail.com

Owner@OWNER-PC ~
$
```

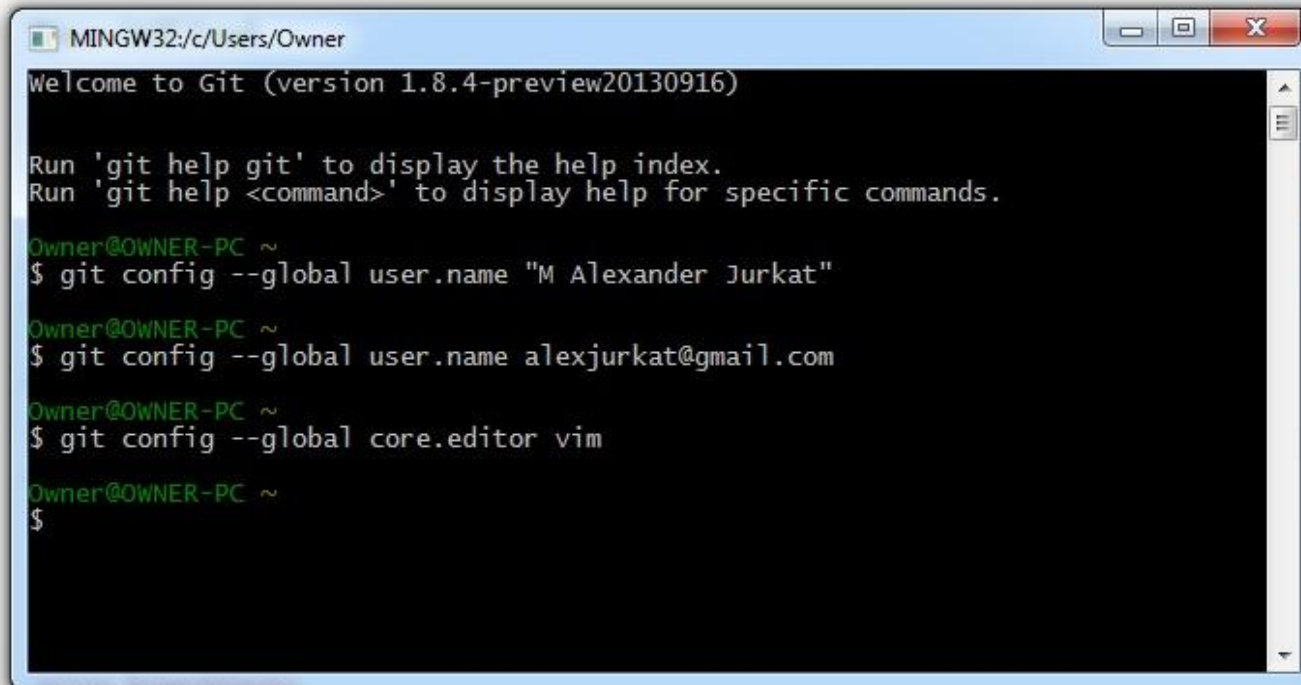
Git needs to know who you are so it can sign all your contributions. That way everyone knows who you are and what you did.

How Are You Editing?

Your Editor

Now that your identity is set up, you can configure the default text editor that will be used when Git needs you to type in a message. By default, Git uses your system's default editor, which is generally Vi or Vim. If you want to use a different text editor, such as Emacs, you can do the following:

```
$ git config --global core.editor emacs
```



```
MINGW32:/c/Users/Owner
Welcome to Git (version 1.8.4-preview20130916)

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

Owner@OWNER-PC ~
$ git config --global user.name "M Alexander Jurkat"

Owner@OWNER-PC ~
$ git config --global user.name alexjurkat@gmail.com

Owner@OWNER-PC ~
$ git config --global core.editor vim

Owner@OWNER-PC ~
$
```

Git needs to know which command line editor you prefer. Pro Git uses the Emacs editor, but we will be using Vim (see screen capture).

How Will You Resolve Merge Conflicts?

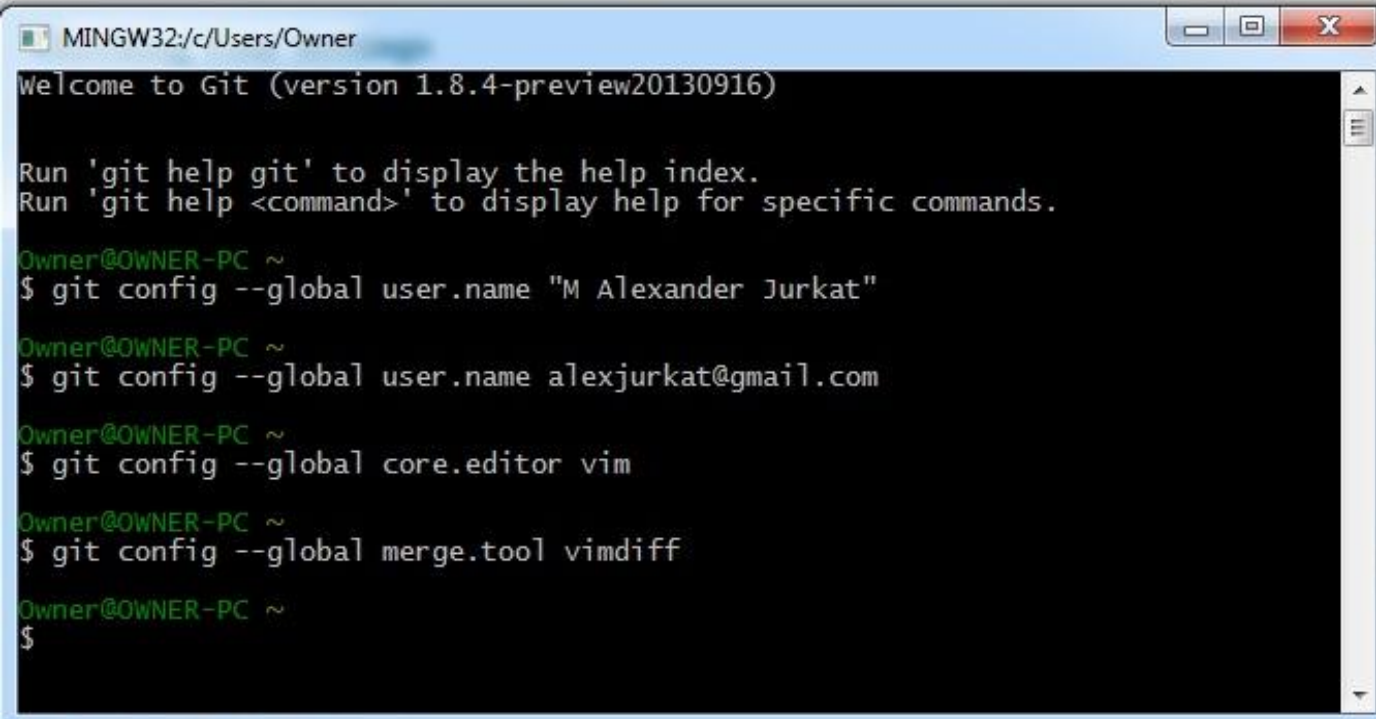
Your Diff Tool

Another useful option you may want to configure is the default diff tool to use to resolve merge conflicts. Say you want to use vimdiff:

```
$ git config --global merge.tool vimdiff
```

Git accepts kdiff3, tkdiff, meld, xxdiff, emerge, vimdiff, gvimdiff, ecmerge, and opendiff as valid merge tools. You can also set up a custom tool; see Chapter 7 for more information about doing that.

We'll get to merge conflicts later. They occur when two people try to change the same line of the same file at the same time -- the hazards of collaboration.



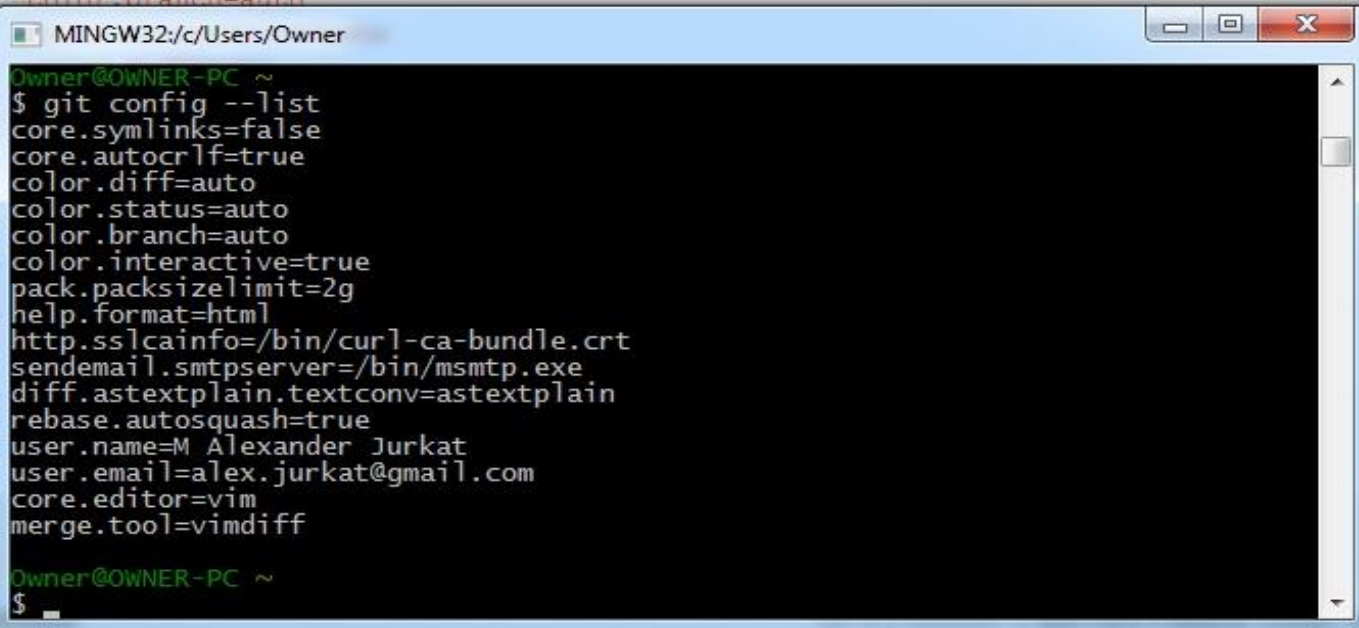
```
MINGW32/c/Users/Owner
Welcome to Git (version 1.8.4-preview20130916)
Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.
Owner@OWNER-PC ~
$ git config --global user.name "M Alexander Jurkat"
Owner@OWNER-PC ~
$ git config --global user.name alexjurkat@gmail.com
Owner@OWNER-PC ~
$ git config --global core.editor vim
Owner@OWNER-PC ~
$ git config --global merge.tool vimdiff
Owner@OWNER-PC ~
$
```

Check Your Work

Checking Your Settings

If you want to check your settings, you can use the `git config --list` command to list all the settings Git can find at that point:

```
$ git config --list
user.name=Scott Chacon
user.email=schacon@gmail.com
color.status=auto
color.branch=auto
```

A screenshot of a Windows command prompt window titled "MINGW32:/c/Users/Owner". The prompt shows the command "\$ git config --list" and its output, which lists various Git configuration settings for a user named "M Alexander Jurkat". The settings include core.symlinks, core.autocrlf, color.diff, color.status, color.branch, color.interactive, pack.packsize, help.format, http.sslcainfo, sendemail.smtpserver, diff.astextplain, rebase.autosquash, user.name, user.email, core.editor, and merge.tool.

```
Owner@OWNER-PC ~
$ git config --list
core.symlinks=false
core.autocrlf=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
pack.packsize=2g
help.format=html
http.sslcainfo=/bin/curl-ca-bundle.crt
sendemail.smtpserver=/bin/msmtp.exe
diff.astextplain.textconv=astextplain
rebase.autosquash=true
user.name=M Alexander Jurkat
user.email=alex.jurkat@gmail.com
core.editor=vim
merge.tool=vimdiff

Owner@OWNER-PC ~
$
```

By reviewing your configuration settings, you can see if how you've done. Correct anything that seems wrong.

Git Help

- Review [Section 1.6 Getting Help](#).
- Play around with this if you like, but this command is for later reference.

Getting Started -- Done!

**Time to
SMILE!**

Git Basics

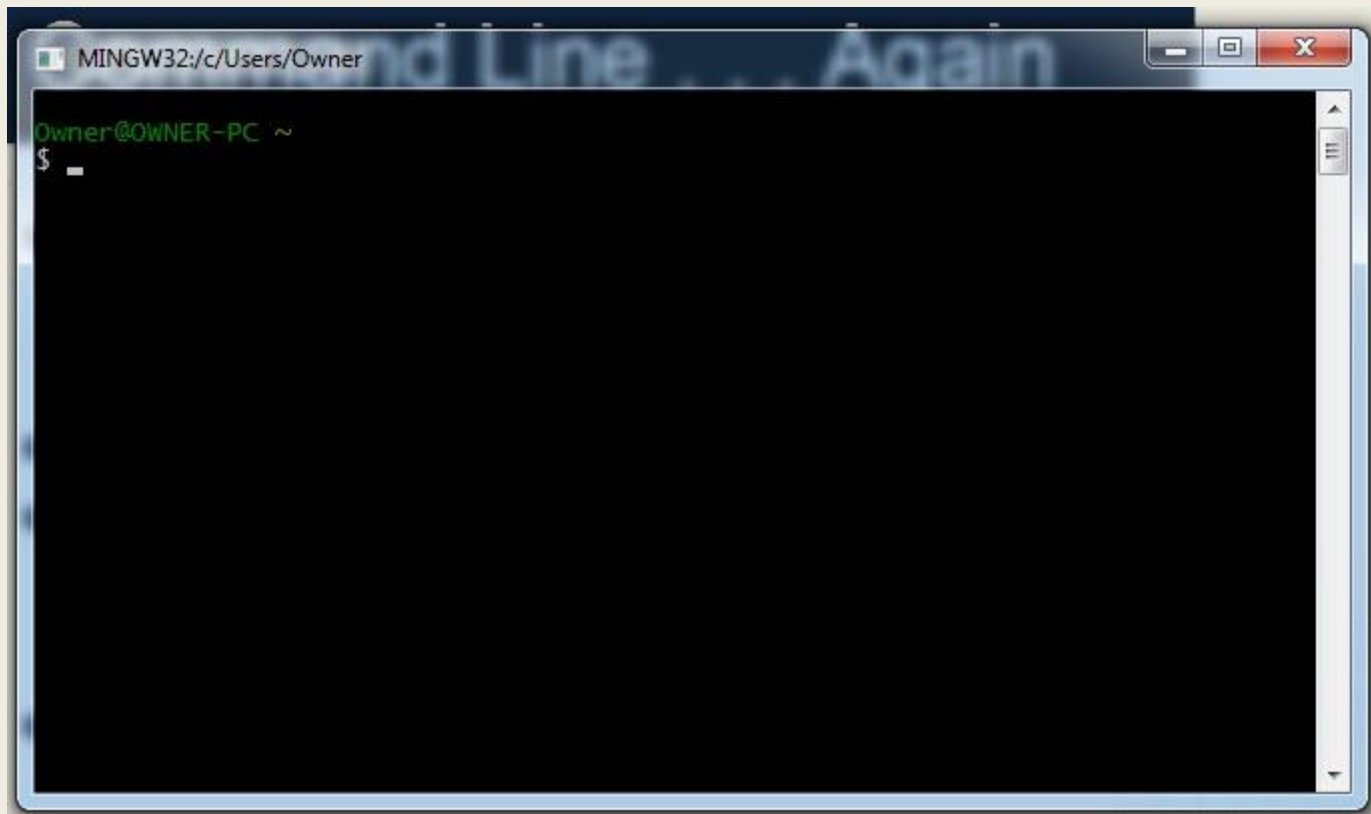
- Review [Chapter 2 Git Basics](#).
- Lots of ground to cover here.

Let's get started!

Command Line . . . Again

- The Git Bash window that you've opened uses the same commands you used for your Command Line assignment.
- Here's a new one: `clear`.
- Type "`clear`" to clean up your window. You should be at the top with only a command prompt.
- Much neater, no?

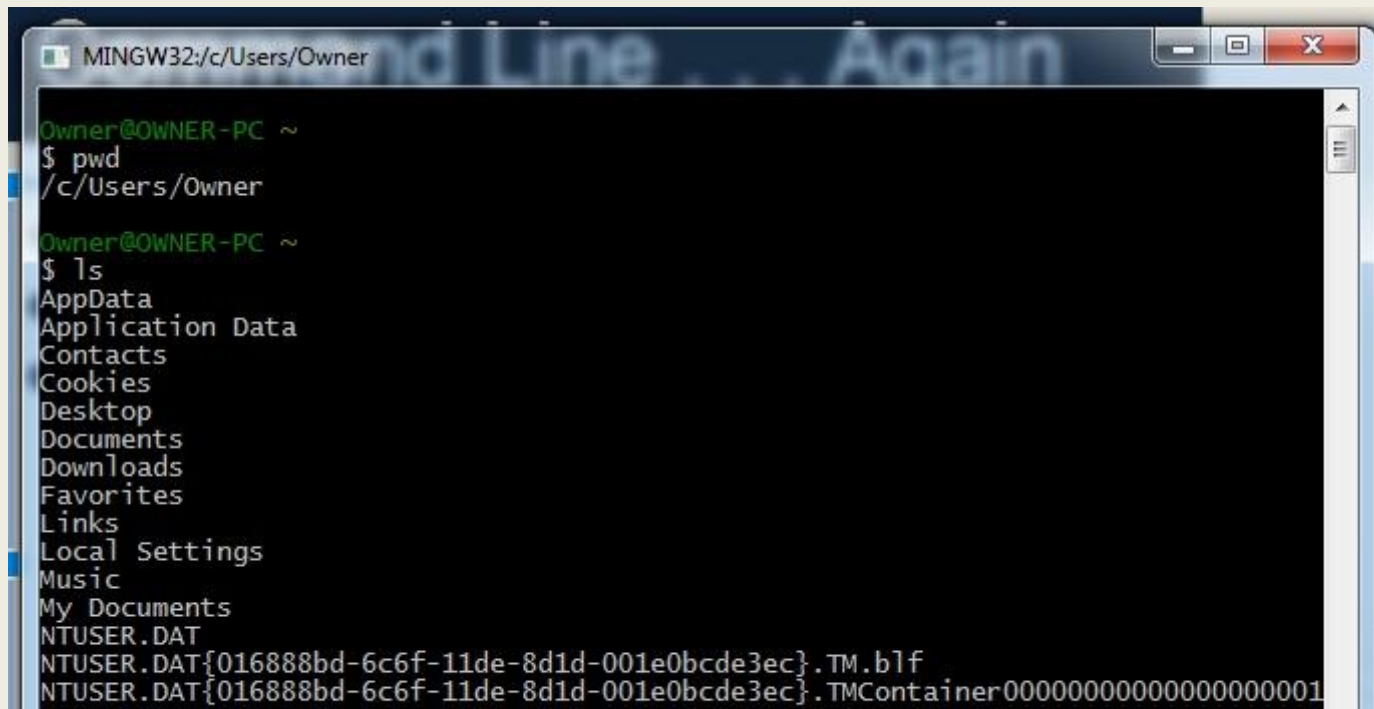
Results of Clear Command



```
MINGW32:/c/Users/Owner  
Owner@OWNER-PC ~  
$ _
```


Let's Review

- Type “pwd” to see where you are.
- Type “ls” to see what's here. Could be a long list. Scroll up to see the commands.

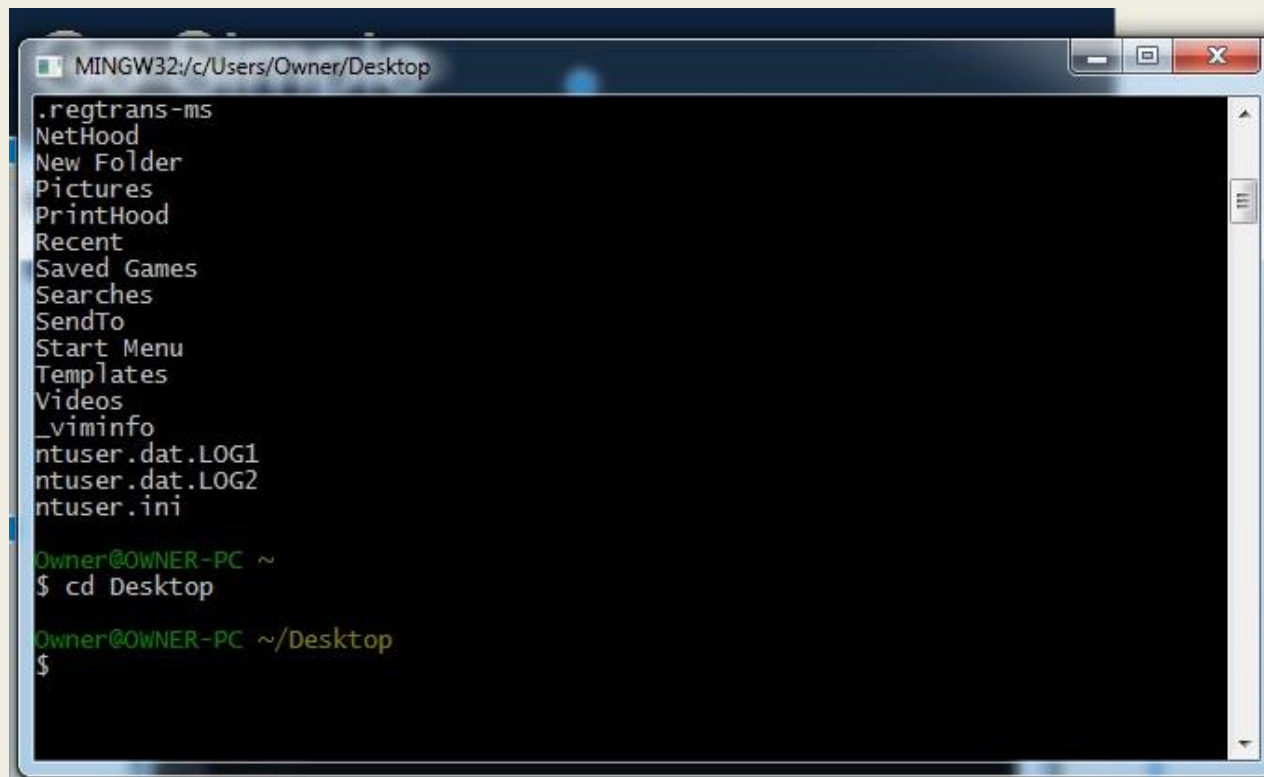


A screenshot of a Windows command prompt window titled "MINGW32:/c/Users/Owner". The window shows the following text:

```
Owner@OWNER-PC ~  
$ pwd  
/c/Users/Owner  
  
Owner@OWNER-PC ~  
$ ls  
AppData  
Application Data  
Contacts  
Cookies  
Desktop  
Documents  
Downloads  
Favorites  
Links  
Local Settings  
Music  
My Documents  
NTUSER.DAT  
NTUSER.DAT{016888bd-6c6f-11de-8d1d-001e0bcde3ec}.TM.blf  
NTUSER.DAT{016888bd-6c6f-11de-8d1d-001e0bcde3ec}.TMContainer0000000000000000000001
```

Go Simple

- Let's move to the Desktop.
- Scroll back to the prompt (if needed) and type “cd Desktop”.



```
MINGW32/c/Users/Owner/Desktop

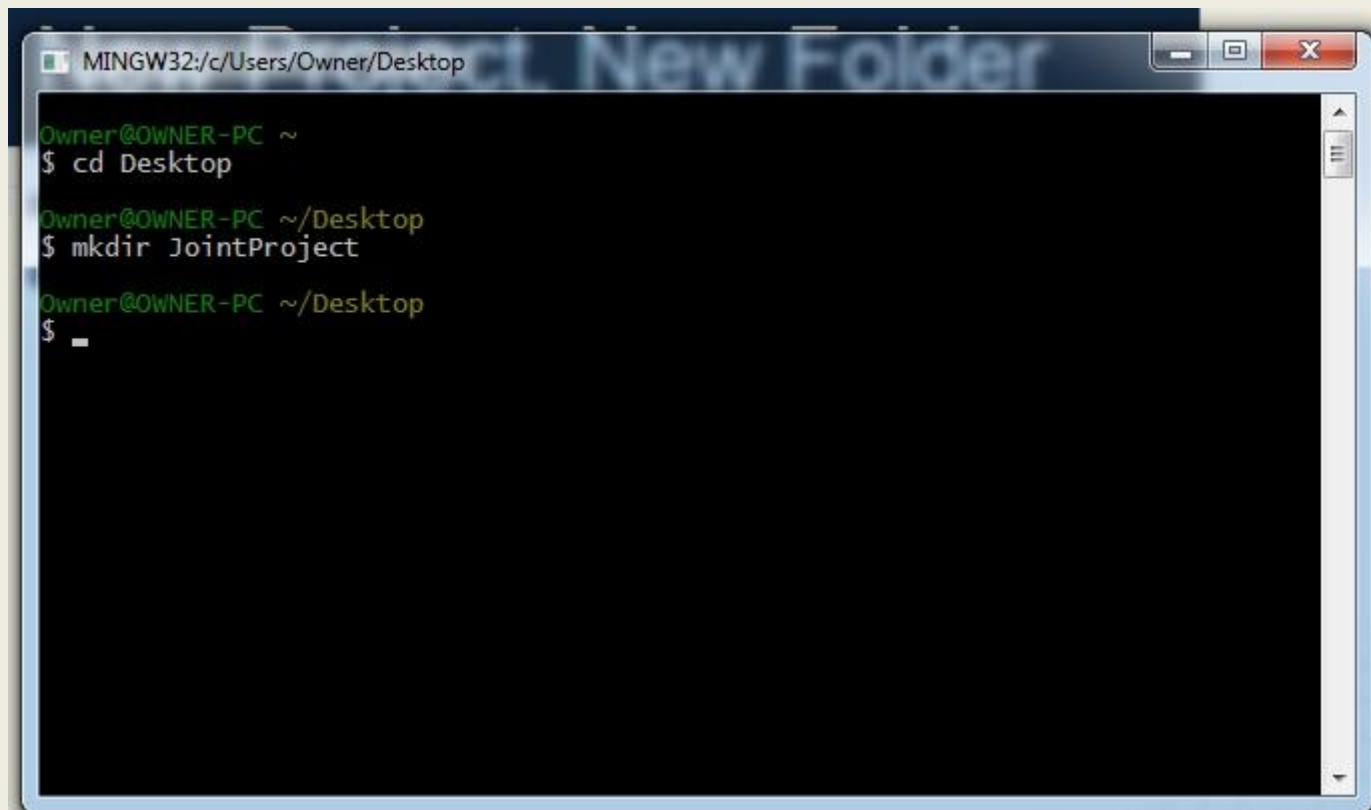
.regtrans-ms
NetHood
New Folder
Pictures
PrintHood
Recent
Saved Games
Searches
SendTo
Start Menu
Templates
Videos
_viminfo
ntuser.dat.LOG1
ntuser.dat.LOG2
ntuser.ini

Owner@OWNER-PC ~
$ cd Desktop

Owner@OWNER-PC ~/Desktop
$
```

New Project, New Folder

- We need a spot for our work.
- Type “mkdir JointProject”.

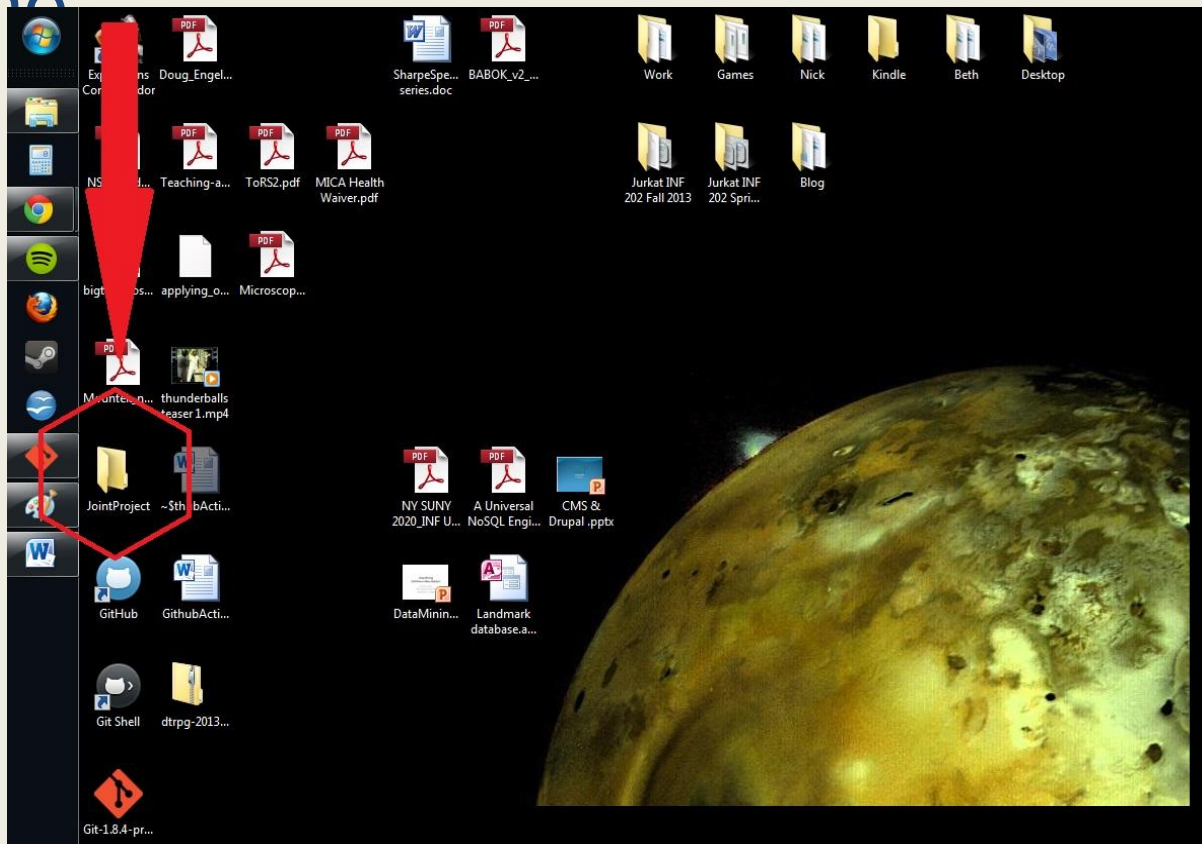


```
MINGW32:/c/Users/Owner/Desktop
Owner@OWNER-PC ~
$ cd Desktop
Owner@OWNER-PC ~/Desktop
$ mkdir JointProject
Owner@OWNER-PC ~/Desktop
$ _
```

The screenshot shows a terminal window titled "MINGW32:/c/Users/Owner/Desktop". The prompt is "Owner@OWNER-PC ~". The user enters "\$ cd Desktop", and the prompt changes to "Owner@OWNER-PC ~/Desktop". The user then enters "\$ mkdir JointProject", and the prompt returns to "Owner@OWNER-PC ~/Desktop". Finally, the user enters "\$ _", and the prompt remains "Owner@OWNER-PC ~/Desktop".

New Project, New Folder

- Check your Windows or Mac Desktop.
- Look at you: Creating using the Command Line

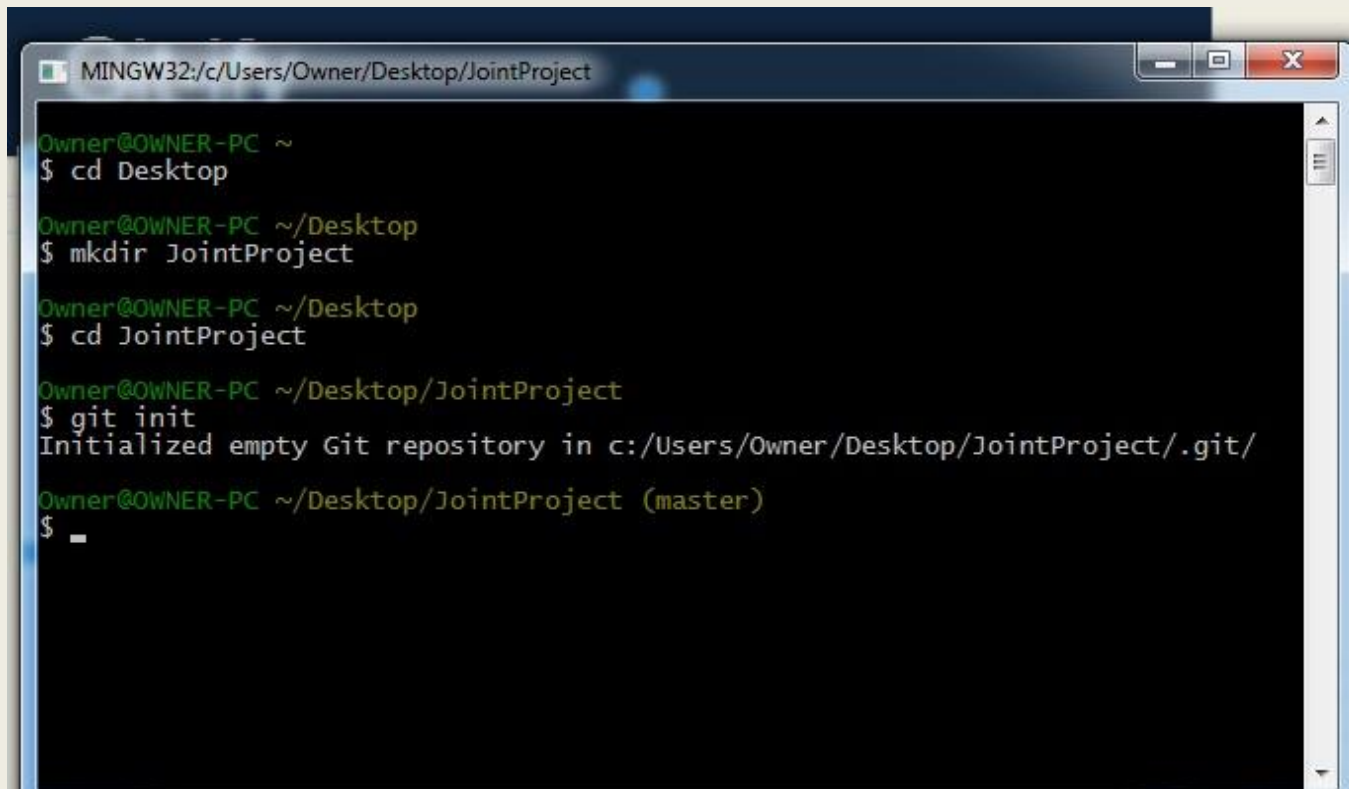


Creating a Git Repository

- Section 2.1 of the Pro Git book details two ways to enter a Git repository -- the place where you store your Git files and the files you are working on.
 - Add existing files and folders to a Git repository.
 - Clone an existing repository from another server.
- We'll be working with the first method. We'll be creating our own Git repository.
- We'll get to cloning in another stage.

Git-ify

- We've got a folder for our project, but it needs the Git processes and code.
- Enter the project folder and type "git init"



```
MINGW32:/c/Users/Owner/Desktop/JointProject
Owner@OWNER-PC ~
$ cd Desktop

Owner@OWNER-PC ~/Desktop
$ mkdir JointProject

Owner@OWNER-PC ~/Desktop
$ cd JointProject

Owner@OWNER-PC ~/Desktop/JointProject
$ git init
Initialized empty Git repository in c:/Users/Owner/Desktop/JointProject/.git/

Owner@OWNER-PC ~/Desktop/JointProject (master)
$ _
```

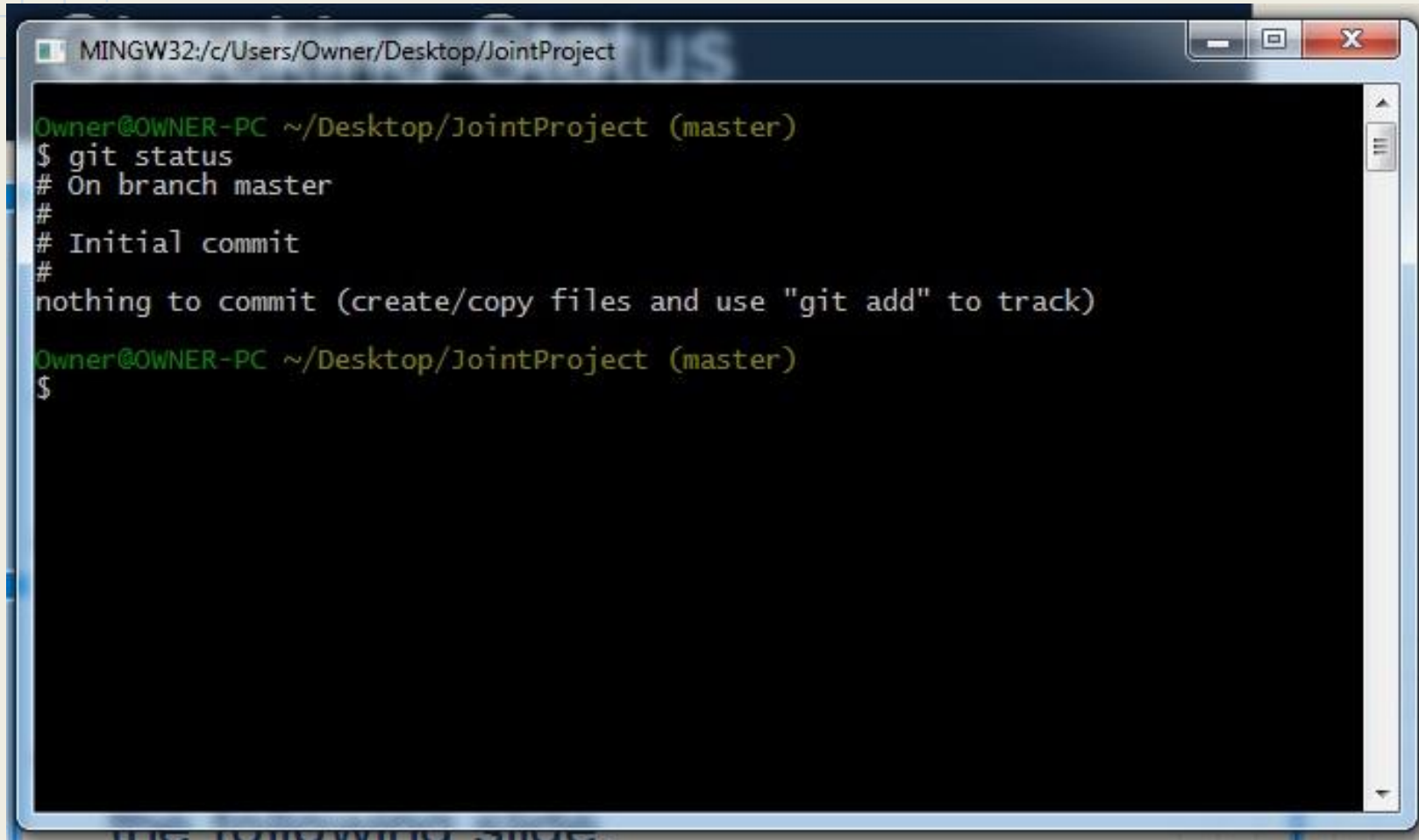
Git-ify

- Git-ifying creates a new folder within JointProject that contains the Git files.
- Because the folder starts with a period, it's hidden. You won't be able to see it by typing "ls" at the command prompt.
- You can show the hidden folder by typing "ls -a".
- You can also see it if you look in the folder on your desktop using Windows Explorer or Finder.

Checking Status

- Review the beginning of Section 2.2 Recording Changes.
- The File Status Lifecycle is a helpful graphic. Keep it in mind.
- Next, check the status of your Git repository by typing “git status” at the command prompt.
- Your screen should look something like the following slide.

Checking Status



```
MINGW32:/c/Users/Owner/Desktop/JointProject

Owner@OWNER-PC ~/Desktop/JointProject (master)
$ git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)

Owner@OWNER-PC ~/Desktop/JointProject (master)
$
```

Checking Status

- Section 2.2 explains nicely.

Checking the Status of Your Files

The main tool you use to determine which files are in which state is the `git status` command. If you run this command directly after a clone, you should see something like this:

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

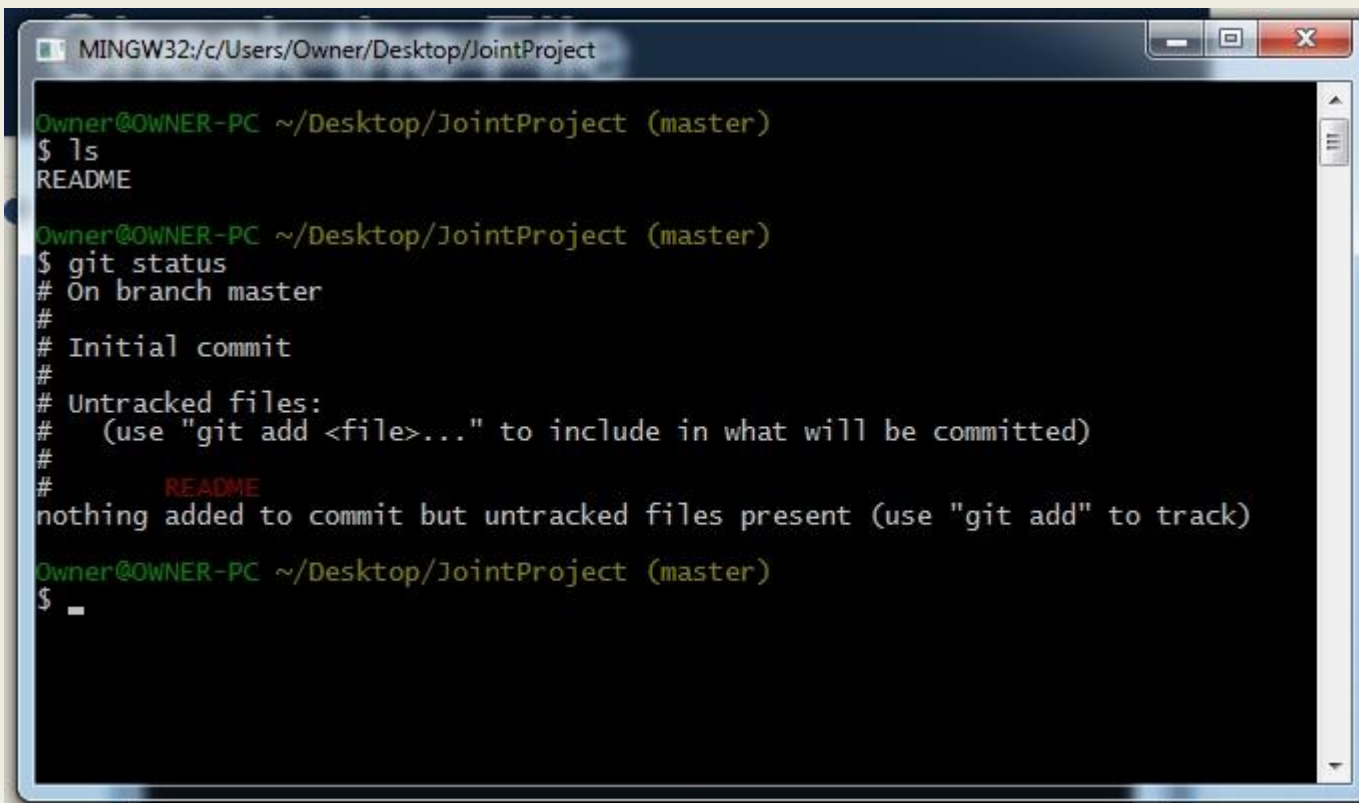
This means you have a clean working directory — in other words, no tracked files are modified. Git also doesn't see any untracked files, or they would be listed here. Finally, the command tells you which branch you're on. For now, that is always `master`, which is the default; you won't worry about it here. The next chapter will go over branches and references in detail.

Add a File

- To add a file to our Git repository, we will use the Vim editor. We used this editor in the Find Students Command Line exercise and when we configured for Git earlier in this lesson.
- Type “vim README”.
- Press “i” to enter insert mode.
- Type whatever you like. I went with “Hello World!”
- Press “Esc” then type “:wq” to save.

Check the File

- Type “ls”. Then “git status”.



```
MINGW32:/c/Users/Owner/Desktop/JointProject

Owner@OWNER-PC ~/Desktop/JointProject (master)
$ ls
README

Owner@OWNER-PC ~/Desktop/JointProject (master)
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       README
nothing added to commit but untracked files present (use "git add" to track)

Owner@OWNER-PC ~/Desktop/JointProject (master)
$
```

Checking Status

- Again, section 2.2 explains nicely.

Let's say you add a new file to your project, a simple `README` file. If the file didn't exist before, and you run `git status`, you see your untracked file like so:

```
$ vim README
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   README
nothing added to commit but untracked files present (use "git add" to track)
```

You can see that your new `README` file is untracked, because it's under the "Untracked files" heading in your status output. Untracked basically means that Git sees a file you didn't have in the previous snapshot (commit); Git won't start including it in your commit snapshots until you explicitly tell it to do so. It does this so you don't accidentally begin including generated binary files or other files that you did not mean to include. You do want to start including `README`, so let's start tracking the file.

Time to Track

Tracking New Files

In order to begin tracking a new file, you use the command `git add`. To begin tracking the `README` file, you can run this:

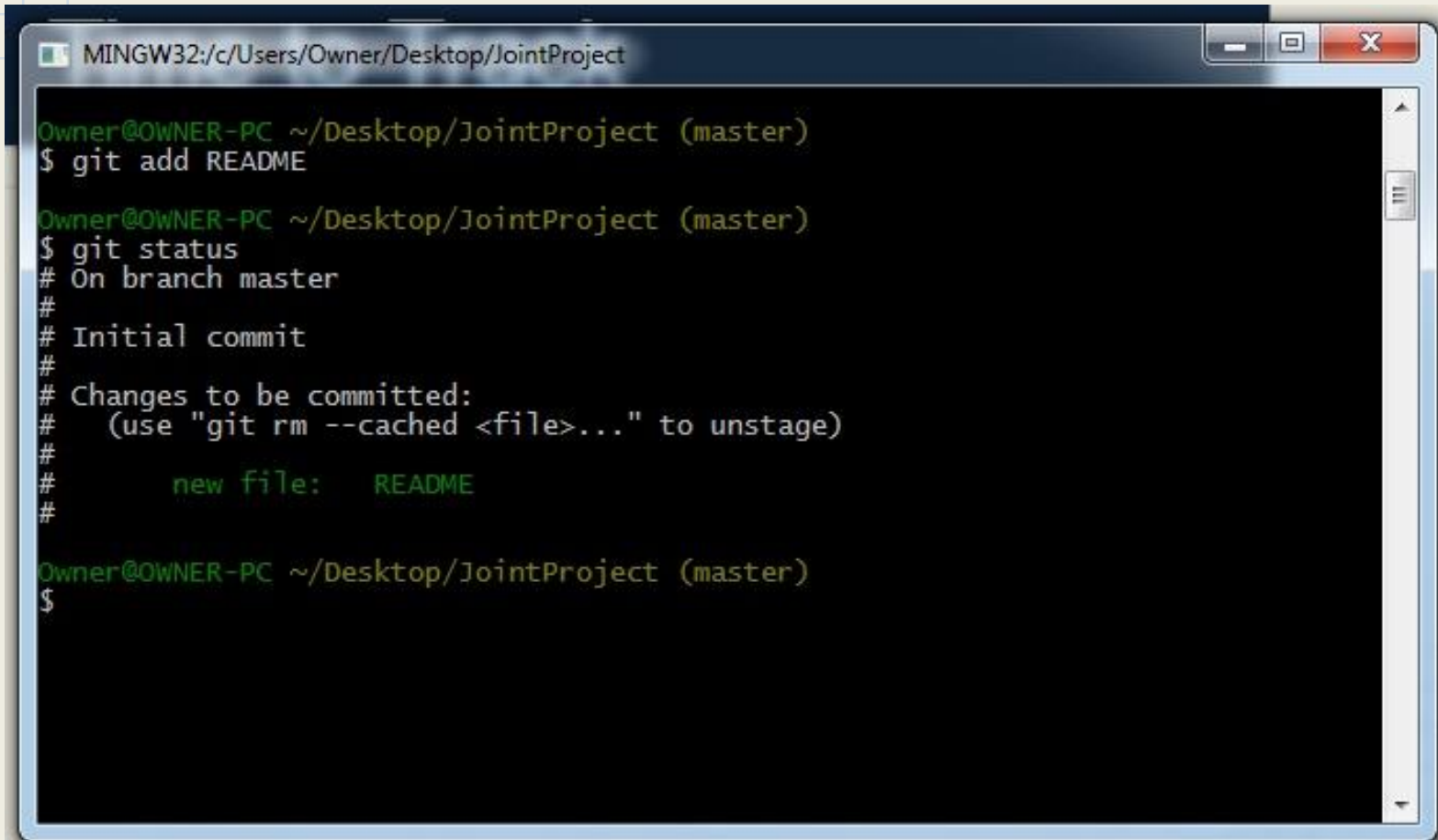
```
$ git add README
```

If you run your status command again, you can see that your `README` file is now tracked and staged:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#
```

You can tell that it's staged because it's under the "Changes to be committed" heading. If you commit at this point, the version of the file at the time you ran `git add` is what will be in the historical snapshot.

Time to Track



```
MINGW32:/c/Users/Owner/Desktop/JointProject

Owner@OWNER-PC ~/Desktop/JointProject (master)
$ git add README

Owner@OWNER-PC ~/Desktop/JointProject (master)
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   README
#
Owner@OWNER-PC ~/Desktop/JointProject (master)
$
```

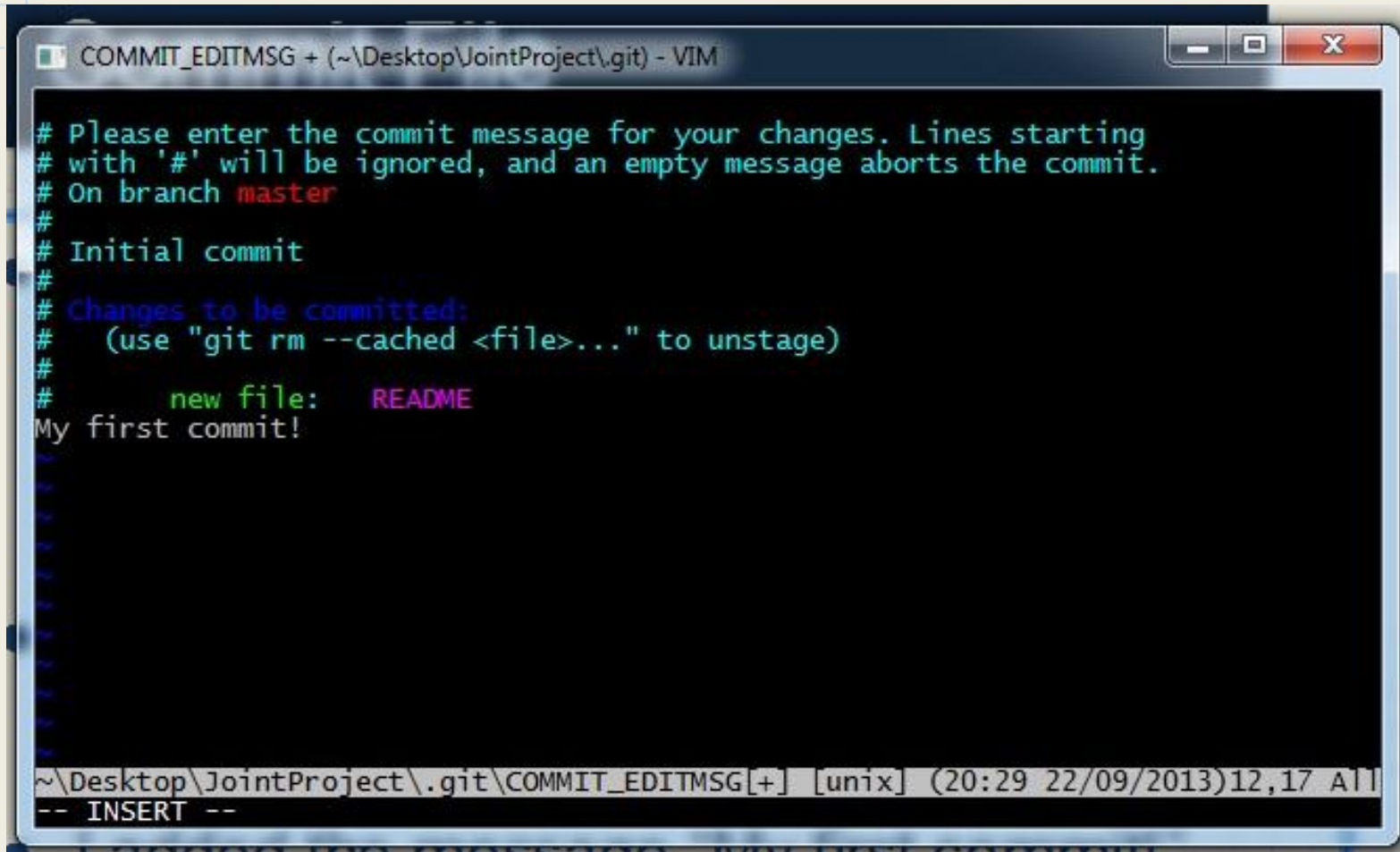
Tracking = Staged

- By tracking the README file, you've set it up or "staged" it to be saved to the repository. That means any changes you made to the file will be included in the next Git snapshot.
- You can go in and make other changes to the README file using "vim README", but if you do so, you need to use "git add README" to make sure you stage those new changes.

Commit File

- Let's assume we're done making changes to the README file (whether you made any or not). Now it's time to create a Git snapshot of the file.
- Type "git commit".
- This launches the Vim editor so you can describe your changes (so collaborators know what you did).

Commit File



```
COMMIT_EDITMSG + (~\Desktop\JointProject\.git) - VIM

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   README
My first commit!

~\Desktop\JointProject\.git\COMMIT_EDITMSG[+] [unix] (20:29 22/09/2013)12,17 All
-- INSERT --
```

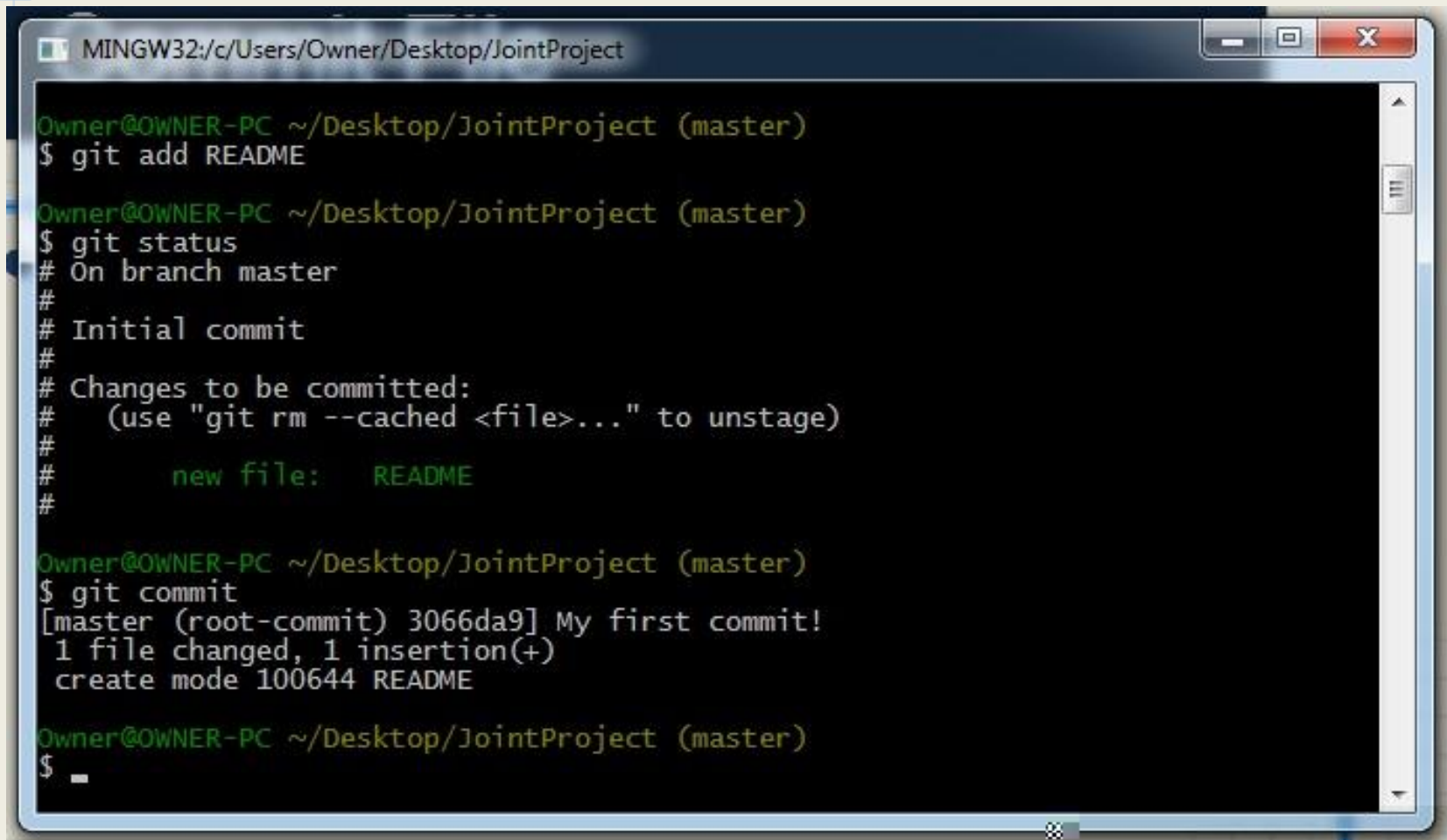
Commit File

- The newly launched Vim editor shows a document with certain “commented” instructions (start with #). Commented instructions will not appear in the commit message.
- I pressed “j” until I got to the bottom of the message, then pressed “i” to enter “insert” mode.
- I added the message “My first commit!”.

Commit File

- Save your message by exiting “insert” mode (Esc) and typing “:wq”.
- You return to the command line (from the Vim editor) and see your first commit.
- The feedback gives you some information about the commit: the branch you committed to (master), the commit’s checksum (in my example, 3066da9), how many files were changed, and lines added or removed.

Commit File



```
MINGW32:/c/Users/Owner/Desktop/JointProject

Owner@OWNER-PC ~/Desktop/JointProject (master)
$ git add README

Owner@OWNER-PC ~/Desktop/JointProject (master)
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   README
#

Owner@OWNER-PC ~/Desktop/JointProject (master)
$ git commit
[master (root-commit) 3066da9] My first commit!
 1 file changed, 1 insertion(+)
 create mode 100644 README

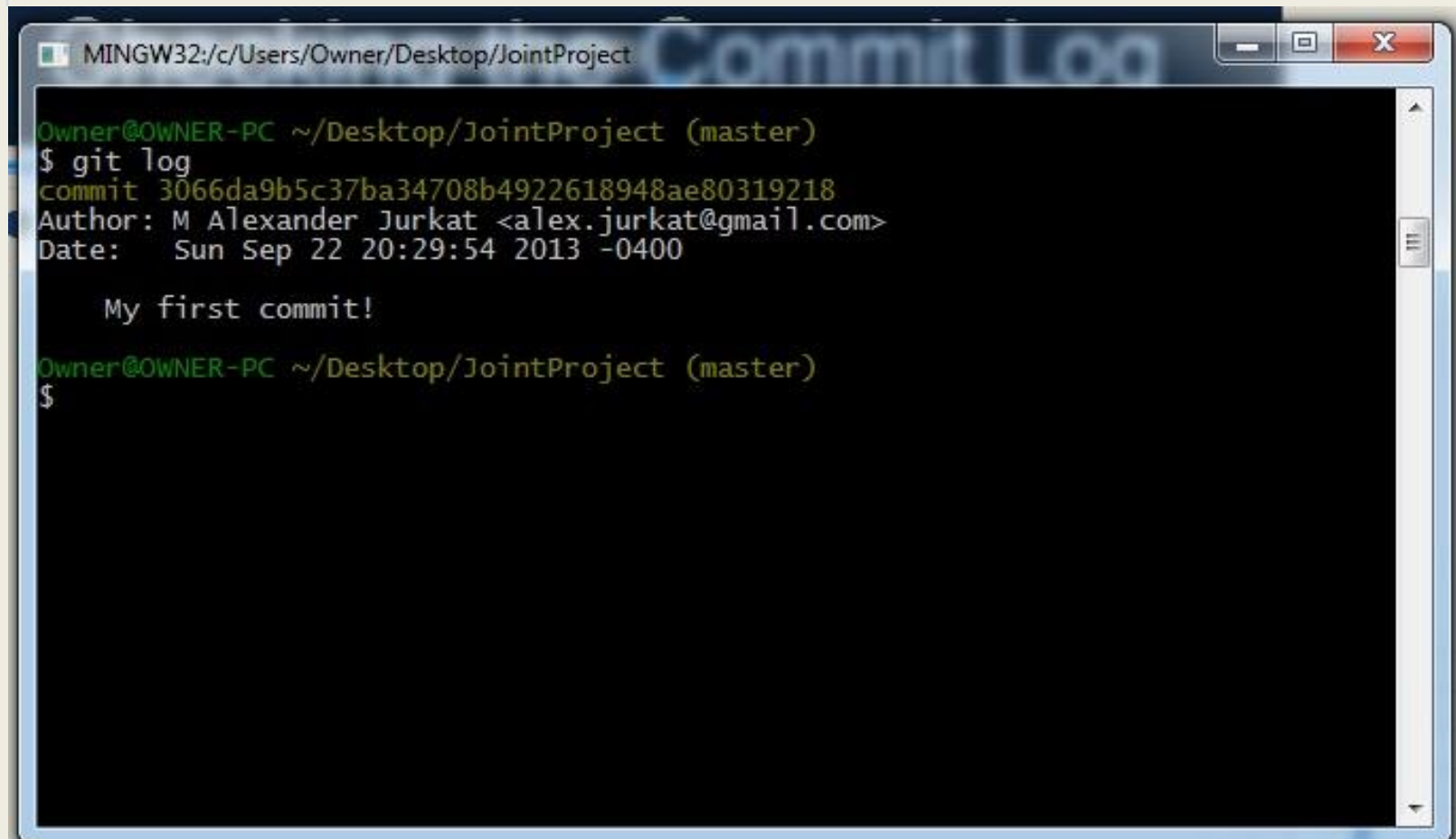
Owner@OWNER-PC ~/Desktop/JointProject (master)
$ _
```

Checking the Commit Log

- You've only made one commit in this exercise, but that's not typical.
- Usually, you make a number of changes and commits. Alternatively, several people work on the same files over time and you want to see what they've changed. To keep track of what you've done, you can check the commit log.
- Type "git log".

Checking the Commit Log

- Type “git log”. There’s your change!

A screenshot of a terminal window titled 'MINGW32:/c/Users/Owner/Desktop/JointProject'. The terminal shows the output of the 'git log' command. The output includes the commit hash '3066da9b5c37ba34708b4922618948ae80319218', the author 'M Alexander Jurkat <alex.jurkat@gmail.com>', the date 'Sun Sep 22 20:29:54 2013 -0400', and the commit message 'My first commit!'. The prompt '\$' is visible at the bottom of the terminal.

```
MINGW32:/c/Users/Owner/Desktop/JointProject

Owner@OWNER-PC ~/Desktop/JointProject (master)
$ git log
commit 3066da9b5c37ba34708b4922618948ae80319218
Author: M Alexander Jurkat <alex.jurkat@gmail.com>
Date:   Sun Sep 22 20:29:54 2013 -0400

    My first commit!

Owner@OWNER-PC ~/Desktop/JointProject (master)
$
```

Submit Your Assignment

- Take a screen grab of your commit log message by pressing “Print Scr”.
- Paste your screen grab into MS Paint or its iOS equivalent.
- Save the screen grab as “[yourname] ModuleAssignment1a”.
- Submit the screen grab as an attachment to your Module Assignment 1a submission.

That's a Wrap!

**Back to the
SMILING!**