

Terminal & Git

Setup Day Lecture

Terminal

First, let's journey into our machine's terminal. The most direct interface we have with our computer.

Terminal - What it's called

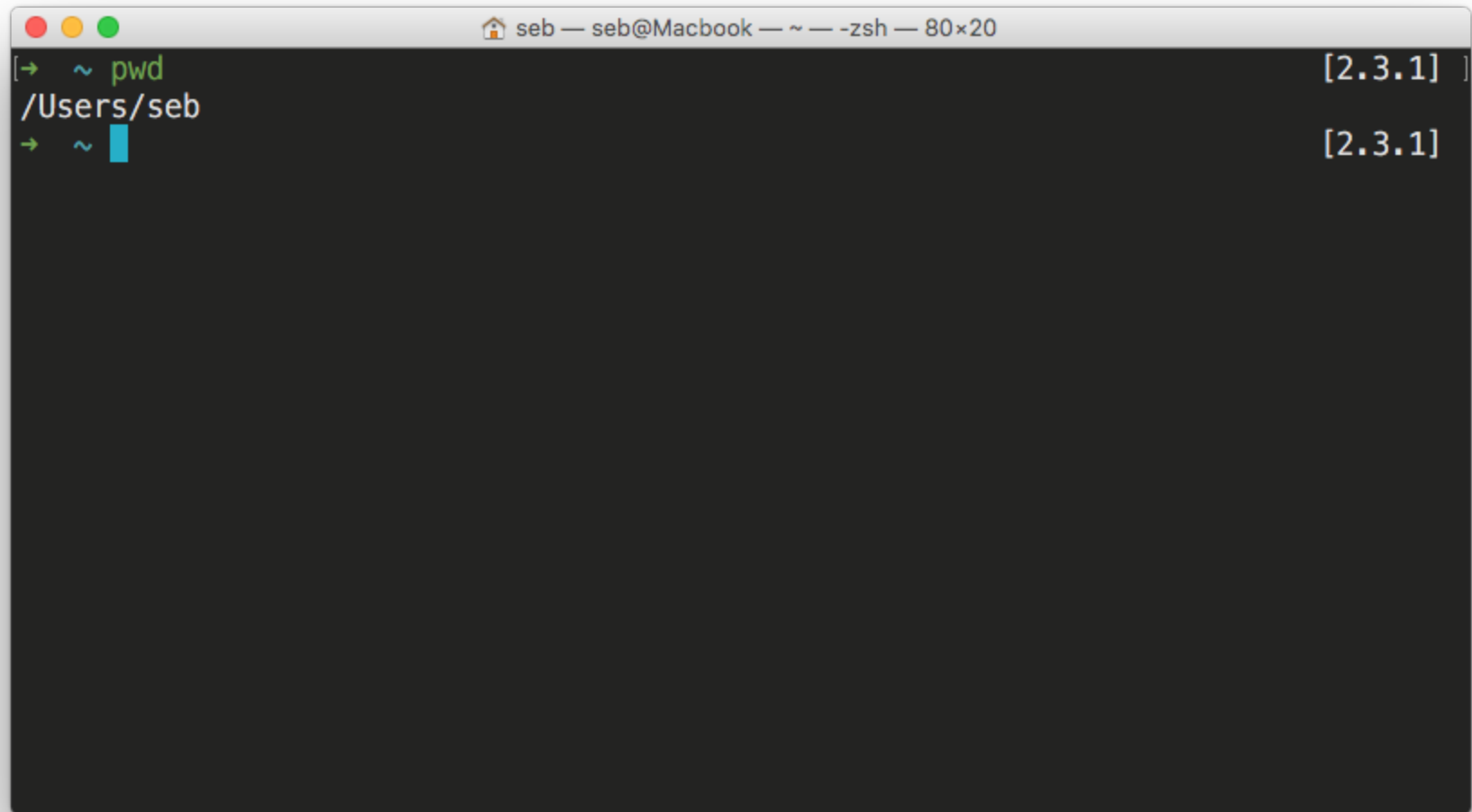
- Command Prompt
- Console
- Terminal

Terminal - Basic Commands

Terminal - Current Location

1. Look for the directory name before the prompt, after →
2. Or print the path of the current directory

```
pwd
```

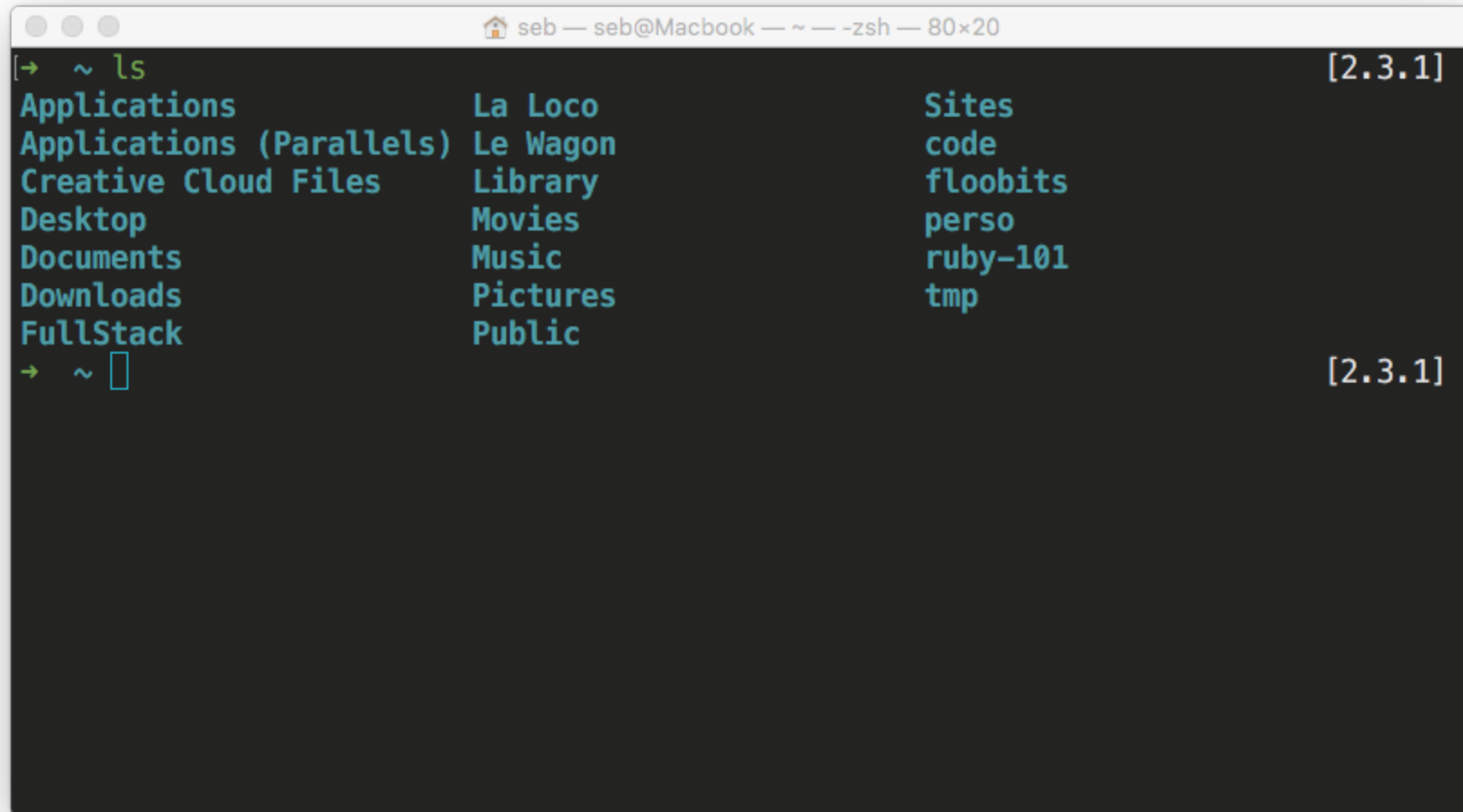


```
seb — seb@Macbook — ~ — -zsh — 80x20  
[→ ~ pwd [2.3.1]  
/Users/seb  
→ ~ [2.3.1]
```

That's your `$HOME` directory.

Terminal - Where Can I Go?

`ls` (or `ll`, an alias of `ls -lh`)

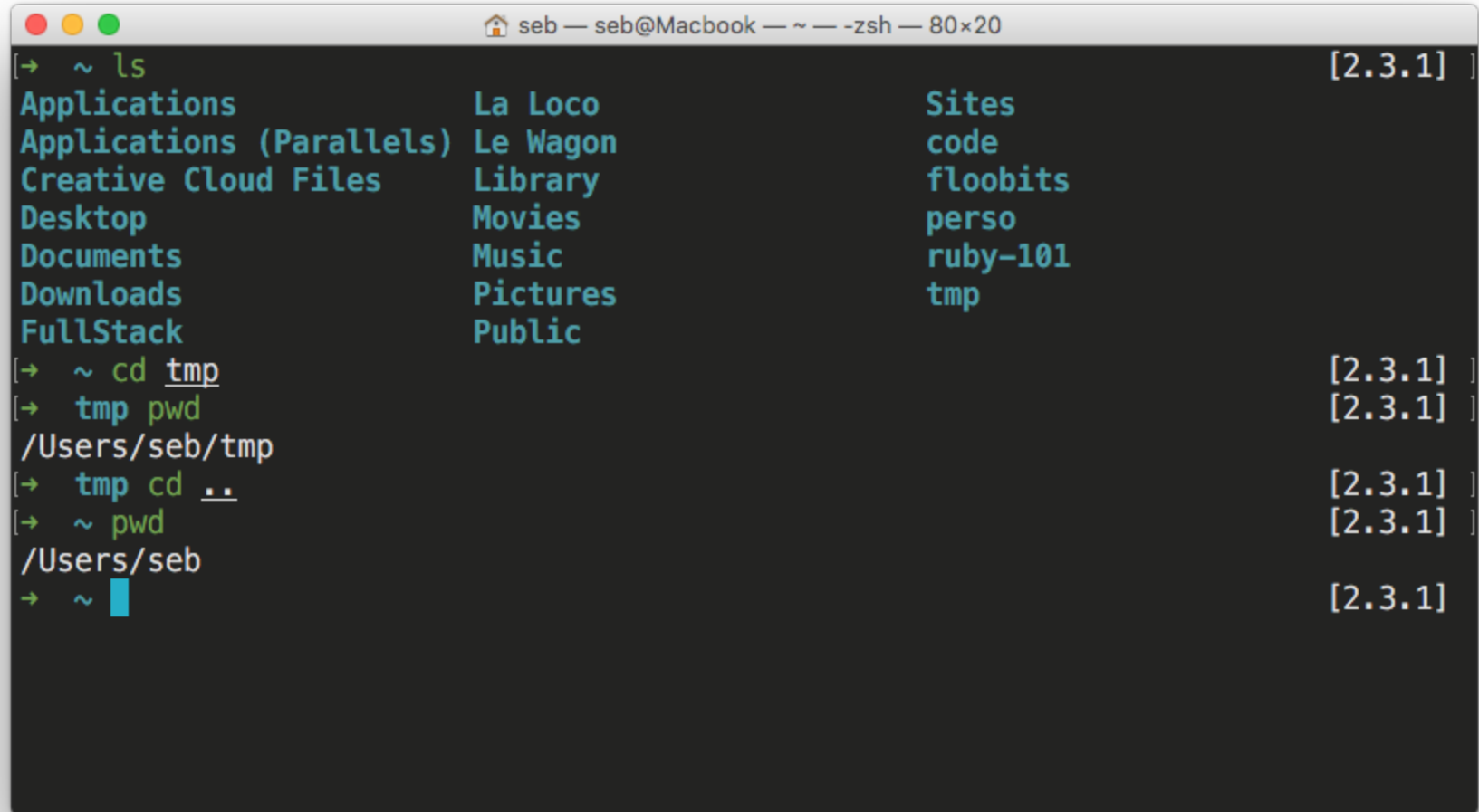


```
seb — seb@Macbook — ~ — zsh — 80x20
[→ ~ ls [2.3.1]
Applications      La Loco           Sites
Applications (Par Le Wagon  code
Creative Cloud Fi Library          floobits
Desktop           Movies            perso
Documents         Music             ruby-101
Downloads         Pictures          tmp
FullStack        Public
[→ ~ [2.3.1]
```

The image shows a terminal window with a dark background and light-colored text. The window title bar at the top reads "seb — seb@Macbook — ~ — zsh — 80x20". The terminal content shows the command `ls` being executed in the home directory (~). The output is a three-column listing of files and directories. The first column contains standard macOS system folders like Applications, Desktop, and Documents. The second column contains user-specific folders like La Loco, Le Wagon, and Library. The third column contains various project or personal folders like Sites, code, and floobits. The prompt `→ ~` is shown at the bottom, indicating the user is ready for a new command.

Terminal - Let's Go There

cd <FOLDER_NAME>



```
seb — seb@Macbook — ~ — zsh — 80x20
[→ ~ ls [2.3.1] ]
Applications                               La Loco                               Sites
Applications (Parallels) Le Wagon                               code
Creative Cloud Files      Library                               floobits
Desktop                   Movies                               perso
Documents                 Music                               ruby-101
Downloads                 Pictures                             tmp
FullStack                 Public
[→ ~ cd tmp [2.3.1] ]
[→ tmp pwd [2.3.1] ]
/Users/seb/tmp
[→ tmp cd .. [2.3.1] ]
[→ ~ pwd [2.3.1] ]
/Users/seb
[→ ~ [2.3.1]
```


Terminal - Ascending a Directory

```
cd ..
```



A terminal window titled "tmp — seb@Macbook — ~/tmp — -zsh — 80x20". The window shows a directory listing of the home directory (~) and subsequent navigation commands. The directory listing is as follows:

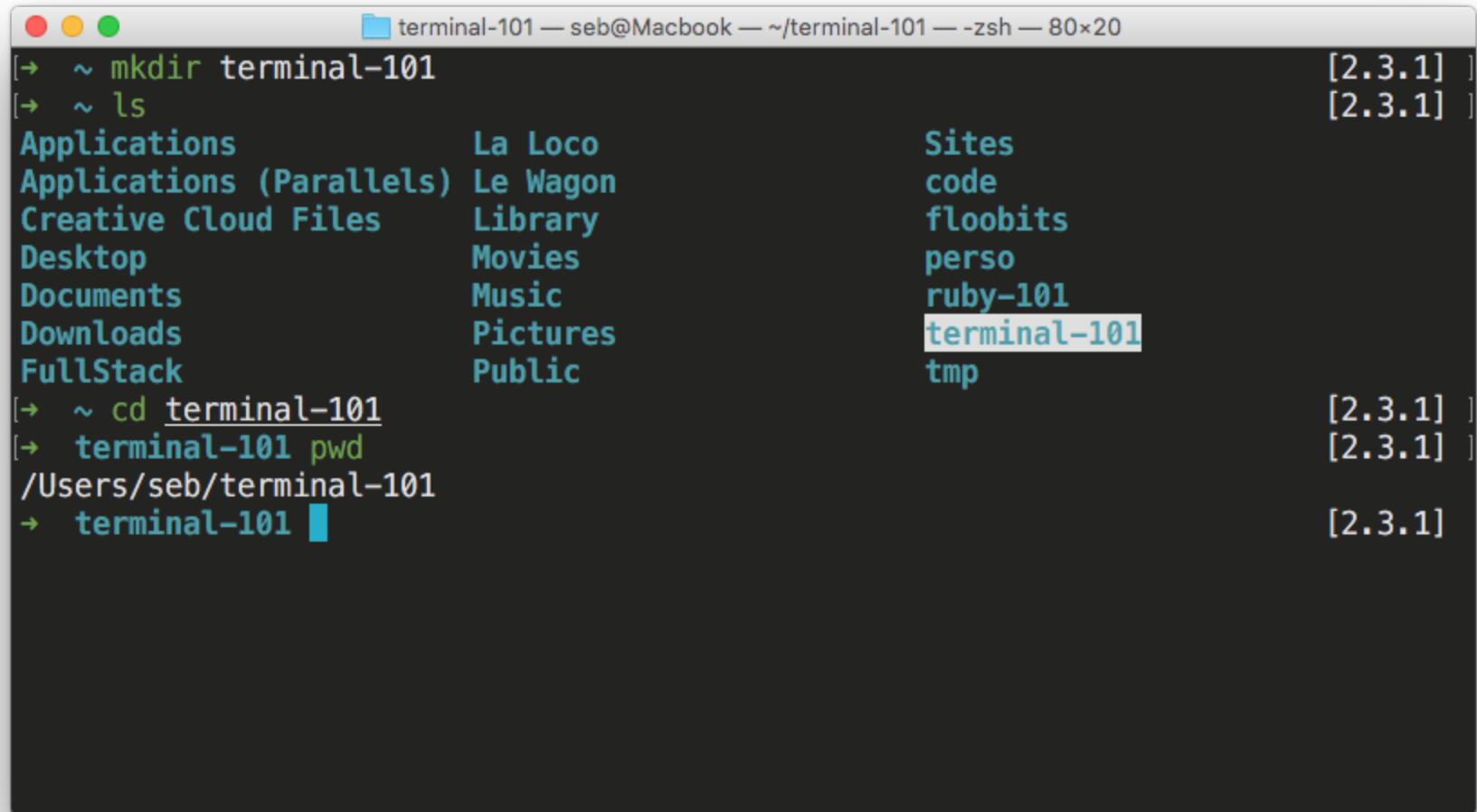
Applications	La Loco	Sites
Applications (Parallels)	Le Wagon	code
Creative Cloud Files	Library	floobits
Desktop	Movies	perso
Documents	Music	ruby-101
Downloads	Pictures	tmp
FullStack	Public	

The terminal shows the following commands and their outputs:

```
[→ ~ ls [2.3.1] ]  
[→ ~ cd tmp [2.3.1] ]  
[→ tmp pwd [2.3.1] ]  
/Users/seb/tmp  
→ tmp [2.3.1]
```

Terminal - Creating a New Directory

```
mkdir <NEW_FOLDER>
```

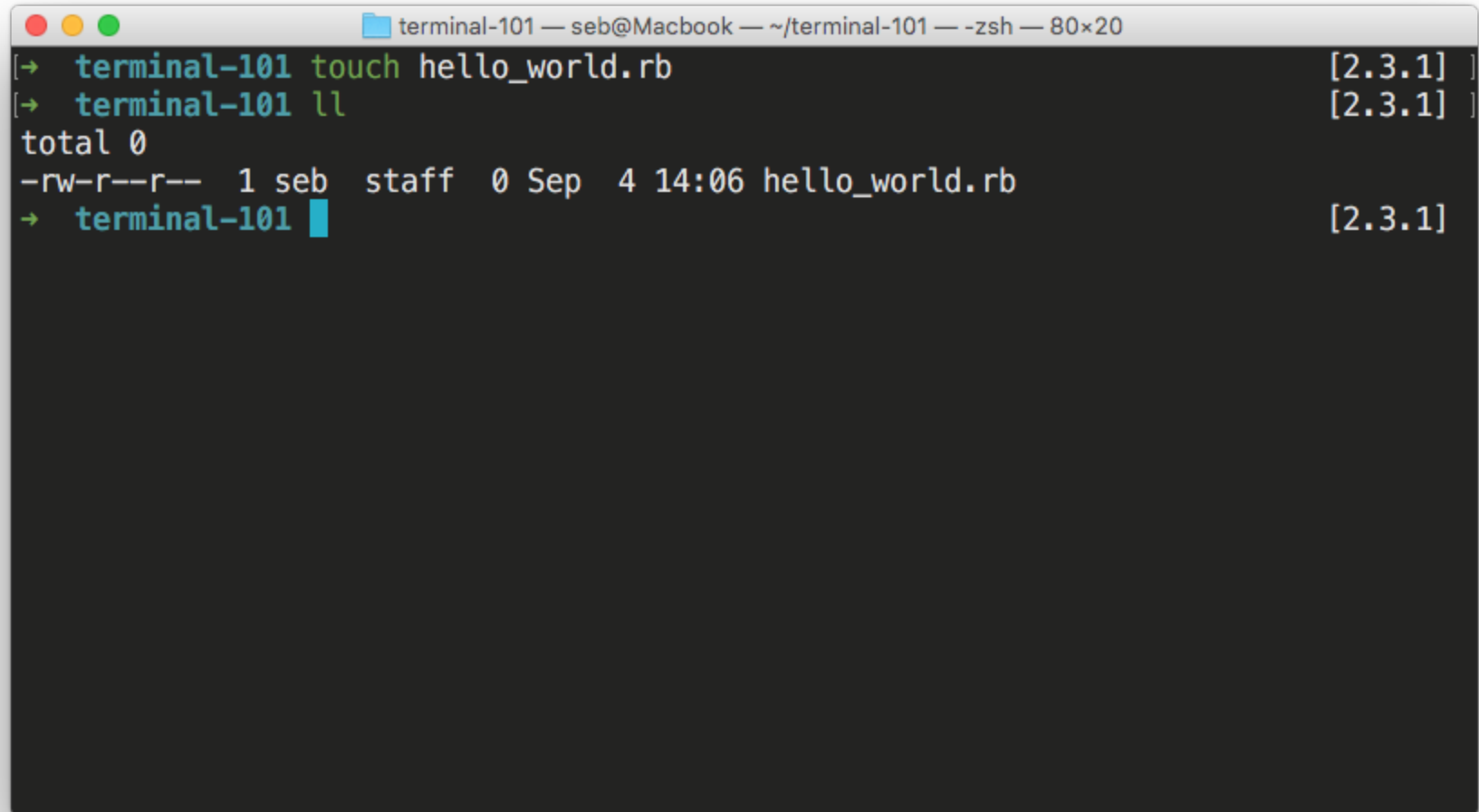


A terminal window titled "terminal-101 — seb@Macbook — ~/terminal-101 — -zsh — 80x20". The window shows the following commands and output:

```
[→ ~ mkdir terminal-101 [2.3.1] ]
[→ ~ ls [2.3.1] ]
Applications          La Loco               Sites
Applications (Parallels) Le Wagon              code
Creative Cloud Files  Library              floobits
Desktop               Movies                perso
Documents             Music                 ruby-101
Downloads             Pictures              terminal-101
FullStack             Public                tmp
[→ ~ cd terminal-101 [2.3.1] ]
[→ terminal-101 pwd [2.3.1] ]
/Users/seb/terminal-101
[→ terminal-101 █ [2.3.1]
```

Terminal - Creating a New File

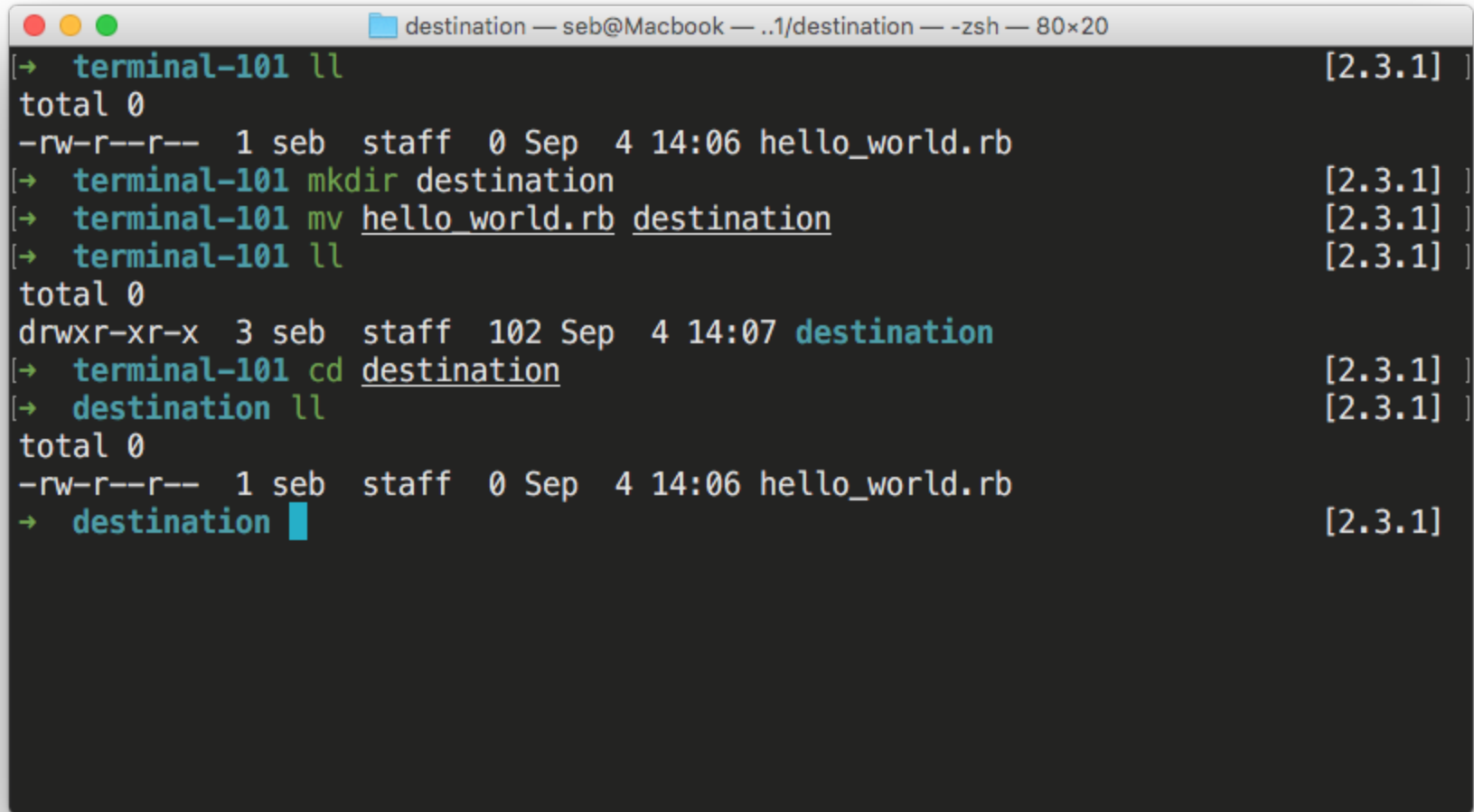
```
touch <FILE_NAME>
```

A terminal window titled "terminal-101 — seb@Macbook — ~/terminal-101 — -zsh — 80x20". The window has a dark background with light-colored text. The user enters the command "touch hello_world.rb" and then "ll". The output shows the file "hello_world.rb" with permissions "-rw-r--r--", owner "seb", group "staff", size "0", and date "Sep 4 14:06". The prompt "terminal-101" is followed by a blue cursor.

```
terminal-101 touch hello_world.rb [2.3.1]
terminal-101 ll [2.3.1]
total 0
-rw-r--r--  1 seb  staff  0 Sep  4 14:06 hello_world.rb
terminal-101 [2.3.1]
```

Terminal - Move Files & Directories

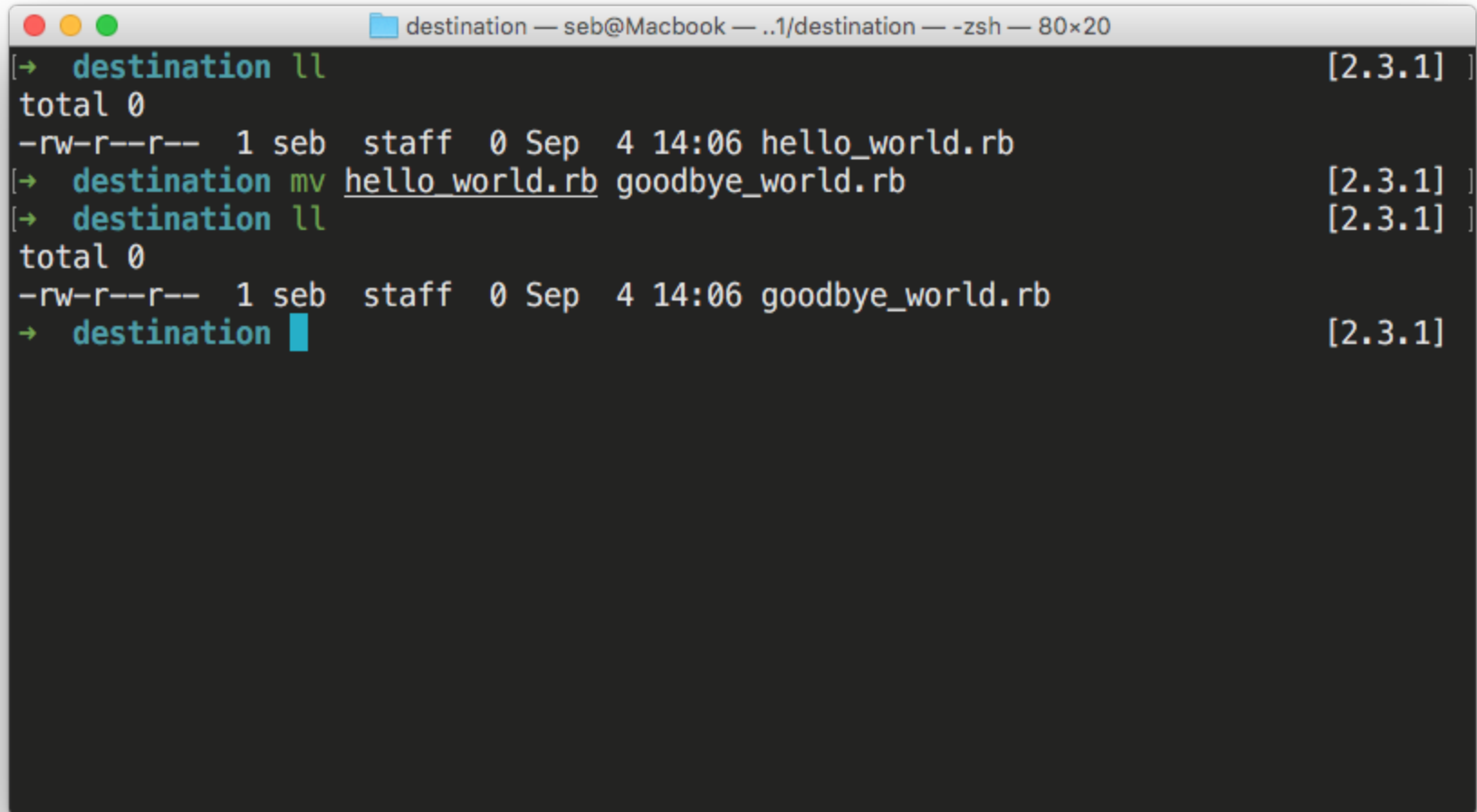
```
mv <FILE_NAME> <FOLDER_NAME>
```

A terminal window titled "destination — seb@Macbook — ../destination — -zsh — 80x20" with a dark background and light-colored text. The window shows a sequence of commands and their outputs. The first command is 'll', which lists the contents of the current directory, showing a file named 'hello_world.rb'. The second command is 'mkdir destination', which creates a new directory named 'destination'. The third command is 'mv hello_world.rb destination', which moves the file 'hello_world.rb' into the 'destination' directory. The fourth command is 'll', which lists the contents of the current directory, showing the newly created 'destination' directory. The fifth command is 'cd destination', which changes the current directory to 'destination'. The sixth command is 'll', which lists the contents of the current directory, showing the file 'hello_world.rb'. The seventh command is 'destination', which is a prompt for the next command.

```
[→ terminal-101 ll [2.3.1] ]
total 0
-rw-r--r--  1 seb  staff   0 Sep  4 14:06 hello_world.rb
[→ terminal-101 mkdir destination [2.3.1] ]
[→ terminal-101 mv hello_world.rb destination [2.3.1] ]
[→ terminal-101 ll [2.3.1] ]
total 0
drwxr-xr-x  3 seb  staff  102 Sep  4 14:07 destination
[→ terminal-101 cd destination [2.3.1] ]
[→ destination ll [2.3.1] ]
total 0
-rw-r--r--  1 seb  staff   0 Sep  4 14:06 hello_world.rb
→ destination [2.3.1]
```

Terminal - Renaming Files & Directories

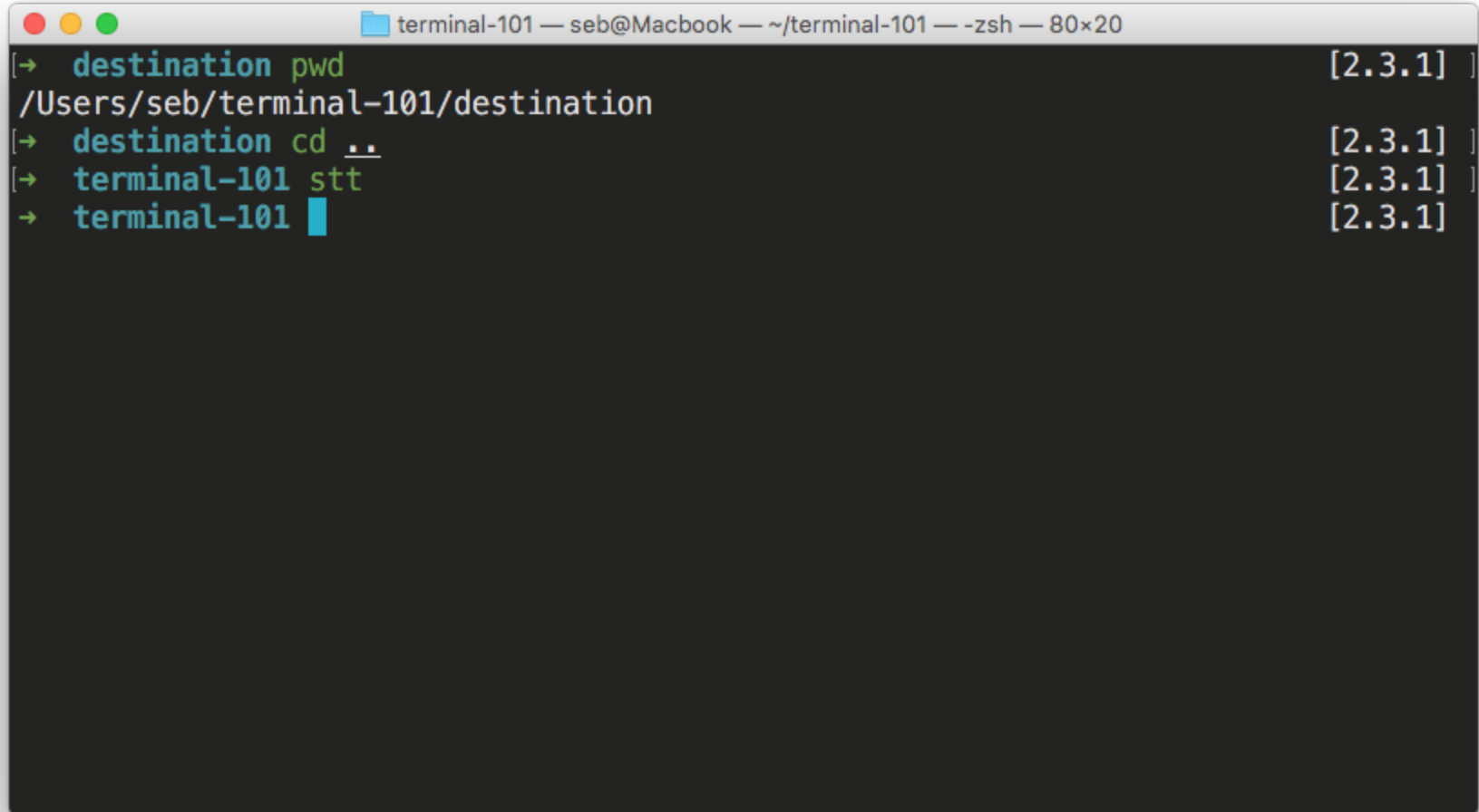
```
mv <FILE_NAME> <NEW_FILENAME>
```

A terminal window titled "destination — seb@Macbook — ../destination — -zsh — 80x20" with a dark background. It shows a sequence of commands and their outputs. First, the user runs "ll", showing a directory listing with "hello_world.rb". Then, they run "mv hello_world.rb goodbye_world.rb". Finally, they run "ll" again, showing the directory listing with "goodbye_world.rb". The prompt "destination" is shown at the end of the last line.

```
destination ll [2.3.1]
total 0
-rw-r--r--  1 seb  staff  0 Sep  4 14:06 hello_world.rb
destination mv hello_world.rb goodbye_world.rb [2.3.1]
destination ll [2.3.1]
total 0
-rw-r--r--  1 seb  staff  0 Sep  4 14:06 goodbye_world.rb
destination [2.3.1]
```

Terminal - Open Current Directory in Sublime Text

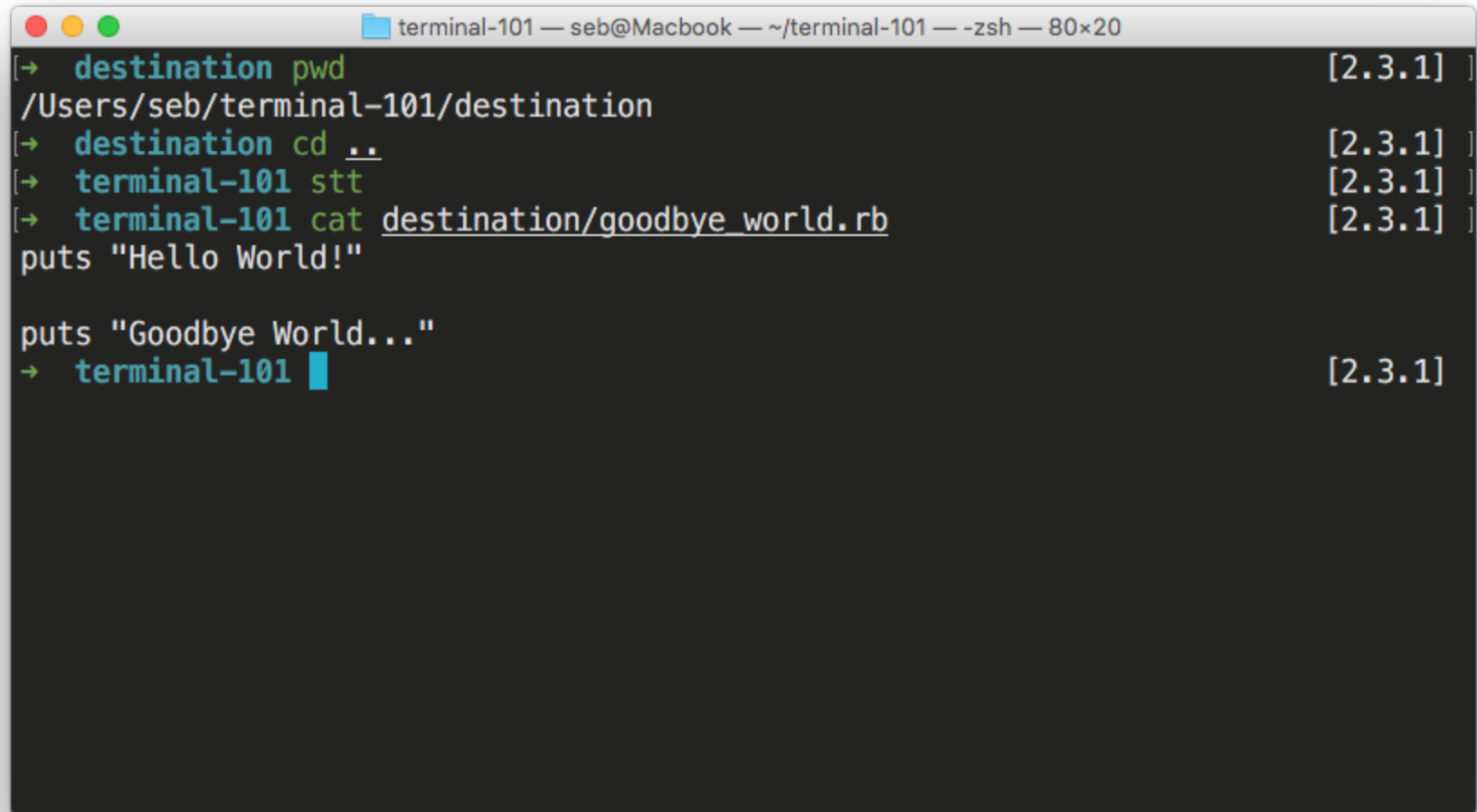
Open your current directory in Sublime with **stt**

A screenshot of a macOS terminal window. The title bar shows 'terminal-101 — seb@Macbook — ~/terminal-101 — -zsh — 80x20'. The terminal has a dark background with light green text. It shows a sequence of commands and their outputs. The first command is 'pwd', which outputs '/Users/seb/terminal-101/destination'. The second command is 'cd ..', which changes the directory to the parent directory. The third command is 'stt', which opens the current directory in Sublime Text. The fourth command is a prompt 'terminal-101' followed by a cursor. The output of the 'stt' command is '[2.3.1]'.

```
[→ destination pwd [2.3.1] ]  
/Users/seb/terminal-101/destination  
[→ destination cd .. [2.3.1] ]  
[→ terminal-101 stt [2.3.1] ]  
→ terminal-101 [2.3.1]
```

Terminal - View Content of File

```
cat <FILE_NAME>
```

A terminal window titled 'terminal-101 — seb@Macbook — ~/terminal-101 — -zsh — 80x20'. The window has a dark background with light-colored text. It shows a series of commands and their outputs. The first command is 'destination pwd', which outputs '/Users/seb/terminal-101/destination'. The second command is 'destination cd ..', which outputs an empty line. The third command is 'terminal-101 stt', which outputs an empty line. The fourth command is 'terminal-101 cat destination/goodbye_world.rb', which outputs 'puts "Hello World!"' and 'puts "Goodbye World..."'. The prompt '→ terminal-101' is followed by a blue cursor. On the right side of the terminal, there are four vertical lines of text: '[2.3.1]', '[2.3.1]', '[2.3.1]', and '[2.3.1]'.

```
terminal-101 — seb@Macbook — ~/terminal-101 — -zsh — 80x20
[→ destination pwd [2.3.1] ]
/Users/seb/terminal-101/destination
[→ destination cd .. [2.3.1] ]
[→ terminal-101 stt [2.3.1] ]
[→ terminal-101 cat destination/goodbye_world.rb [2.3.1] ]
puts "Hello World!"

puts "Goodbye World..."
→ terminal-101 [2.3.1]
```

Terminal - And many more

Check out this [Cheatsheet](#)

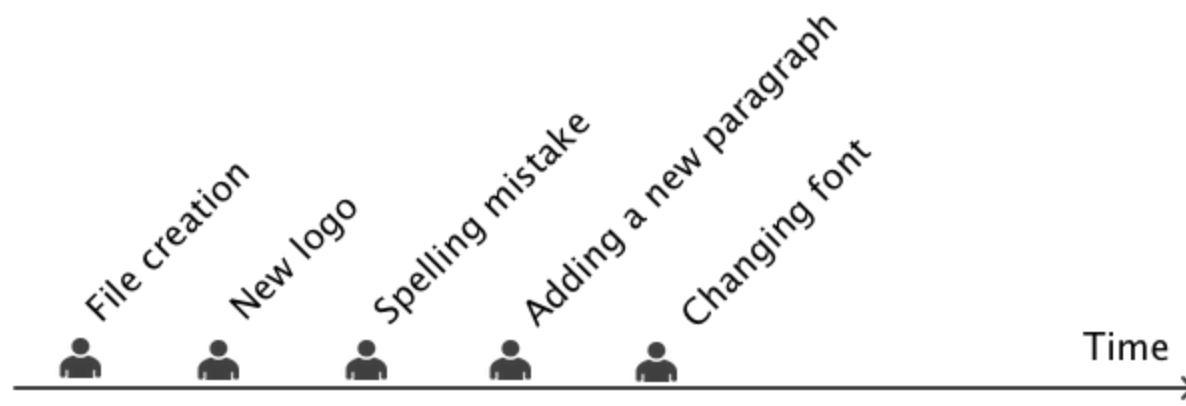
Git

We are Knowledge Workers. We create and edit **files** (text, images, etc.)

Everyday workflow

1. Create a file
2. Save it
3. Edit it
4. Save it again
5. etc.

File life



Manual Version Control

How most people keep track of different versions of a file



Report (Christmas added).doc



Report (final version).doc



Report (John version).doc



Report (REAL FINAL VERSION).doc



Report.doc

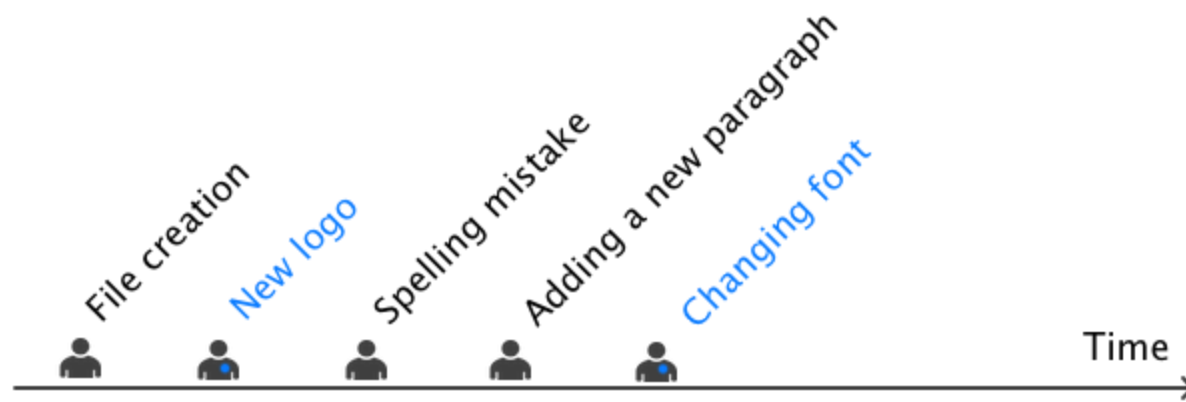
Version Control

This is super annoying and time consuming... Is there anyway we can automate this?

For each document version, we need to know:

1. **When** the file was modified
2. **What** changed
3. **Why** it was modified

There's more: Teams



That's one more question

For each document version, we need to know:

1. When the file was modified
2. What changed
3. Why it was modified
4. **Who** did the change

In a nutshell

We want a tool which:

- tracks document versions
- keeps an history of document changes
- foster team work

That would be



Git Commands

Let's look at some basic commands to get us started

Git - Initializing

```
# From existing repository (on GitHub for instance)  
git clone <github_ssh_clone_url>
```

```
# Or from scratch  
mkdir new_project  
cd new_project  
git init
```

Git - Status

git can tell you if your folder has some modified files (dirty)

```
git status
```

Git - Commit

A `commit` (a snapshot of the folder) is a 3-step job.

```
# First check which files have been modified
git status

# Then, add the ones you want to the staging area.
git add <file_1_which_has_been_modified>
git add <file_2_which_has_been_modified>

# You can review your staging area
git status

# Take a snapshot of what is in the staging area.
git commit --message "A meaningful message about this change"
```

Git - Diff

If `git status` tells you something changed, you can inspect exactly what changed:

```
git diff  
git diff <a_specific_file_or_folder>
```

Git - Log

Show commit history with:

```
git log
```

```
# More fancy command in your ~/.gitconfig  
git lg
```

Git Livecode: Initialize a Repo

Let's create a project and start tracking it

```
mkdir -p ~/code/$GITHUB_USERNAME/git-101  
cd ~/code/$GITHUB_USERNAME/git-101  
git init  
ls -a # it has created a .git hidden folder
```


Git Livecode: 1st Commit

Let's create an `index.html` file and code some basic HTML content

```
touch index.html  
stt  
# code some basic HTML content
```

Time to commit our work

```
git status # file not staged
git add index.html
git status # file staged, ready to commit
git commit -m "Basic HTML content for home page"
git status
```

Git Liveode: 2nd Commit

Let's add an image in our project

```
curl https://raw.githubusercontent.com/lewagon/karr-images/master/white_logo_red_circle.png > logo.png  
stt # add  to your HTML
```

Time to commit our work

```
git status
git diff index.html # what has changed?
git add index.html
git add logo.png
git status
git commit -m "Adding logo to home page"
git status
git log # check commits history
```

Now push it up

```
gh repo create  
gh repo view -w # can you see your new repo on Github?  
git status # check again – anything to be pushed?  
git push origin master
```

Remote Repositories

Code that's stored *remotely*, meaning not on your machine.



Le Wagon

git-101-boilerplate

Simple boilerplate to show how to use git with Github on **@lewagon** setup day.

Updated 10 minutes ago



You




Le Wagon

git-101-boilerplate

Simple boilerplate to show how to use git with Github on @lewagon setup day.

Updated 10 minutes ago

fork 



You

git-101-boilerplate

Simple boilerplate to show how to use git with Github on @lewagon setup day.


Updated 10 minutes ago



Le Wagon

git-101-boilerplate

Simple boilerplate to show how to use git with Github on @lewagon setup day.
Updated 10 minutes ago

fork 



You

git-101-boilerplate

Simple boilerplate to show how to use git with Github on @lewagon setup day.
Updated 10 minutes ago

clone



git-101-boilerplate

Remote - Pushing Changes

Once you've committed your work, push it to Github.

```
# Generic command  
git push <remote> <branch>  
  
# What we'll use  
git push origin master
```

Remote Livecode: Creating a New Repo

1. Let's make a new repo called git-101-practice (like you can for every exercise)
2. Then let's put stuff into it

```
cd ~/code/$GITHUB_USERNAME  
mkdir git-101-practice  
cd git-101-practice  
git init  
git status # it's already tracked by git
```

3. Then make it sync remotely, by adding the github as a remote repo called `origin`

```
git remote add origin git@github.com:lewagon/git-101-practice.git
```

Remote Livecode: Commit and Push

Let's make a change, commit **and push**

```
stt # change the HTML code
git add index.html
git commit -m "adding some custom text"
git status
git push origin master # Pushing on Github
```

Check that **project was updated on Github**.

Git Advanced

In the next few weeks, we'll see how git can help us with

- Solving conflicts
- collaboration (using branches)
- production deployment (using multiple remotes)