Nerevu Handbook

Release 1.0.0

Nerevu Group, LLC

Nov 17, 2021

CONTENTS

CHAPTER

ONE

GETTING SETUP

This section contains a several topics to help a new user get started during their first week at Nerevu.

1.1 System Setup

1.1.1 Initial setup

- Mac OS
- Windows
- Linux

1.1.2 Python environment setup

Install `virtualenv` to manage your Python environments

```
pip install virtualenv
```

Create and activate a virtual environment

```
cd REPO_NAME
virtualenv --no-site-packages --python=python3.x venv
source venv/bin/activate
```

Install required Python libraries

```
pip install -r requirements.txt -r dev-requirements.txt
```

1.1.3 Text Editor setup

- enable Newline at End of File (reason).
- enable Unix-style line endings (`View menu -> Line Endings -> ...` in Sublime Text).
- enable Trim trailing spaces (reason).
- VS Code configuration

1.2 Fastmail

1.2.1 Calendar Sync

Nerevu Group uses Fastmail Calendar to keep everyone up to date on events that are happening. This section will show you how to sync your existing calendars so that they update Fastmail. In general, there are two options for syncing with your teammates:

1. Show them your events with the titles and all details (this is good for a "Work" calendar)
2. Or show them an event block that just indicates that you are busy during that block (no details are shared - this is good for time blocks when you are busy but don't need to share details about why)

We will go over how to setup both of these syncs.

From Google Calendars

See this article on synchronizing Google Calendar with Fastmail here.

1.3 KeePass and Encrypted Volumes

KeePass is the Password Manager that Nerevu Group uses to keep usernames and passwords secure. Encrypted Volumes help us keep important files secure. Please follow the instructions to download both so that you can get access to the Nerevu Group KeePass Database.

1.3.1 Installation

- KeePassXC

1.3.2 Configuration

- open preferences/settings
- click `Security`
- check `Lock Databases after inactivity of` and set seconds to `43200`. This is the maximum time a database can be open before automatically closing. Adding this setting helps us keep our data more secure.
- uncheck `Clear search query after`

1.3.3 Encrypted Volume Configuration

- Mac OS
- Windows
- Linux

1.3.4 KeePass Setup

1. Add keyfile to your *encrypted volume*

- download your `.key` keyfile (link will be given to you)
- cut and paste the downloaded keyfile to your encrypted volume
- make sure the keyfile does not exist anywhere else on your computer except in the encrypted volume (for security reasons)

1. Open Nerevu Group KeePass Database

- Open the KeePass app
- Press `Cancel` if it tries to log into your personal database
- In the menu bar, navigate to `Database > Open database...`
- Navigate to the `.kdbx` password database in dropbox (link will be given to you) and press `Open`
- Enter the password provided to you
- Click `Browse` to navigate the keyfile stored in your encrypted volume
- Click Ok

1. You did it! You accessed the Nerevu Group KeePass Database! Now enjoy a few minutes of you time. You deserve it ;)

1.4 Ngrok

`Ngrok` is a service that allows you to forward localhost to the public web through https. This comes in handy when you're testing code that interacts with an API that requires a valid https URL (like QuickBooks in the Comissioner API).

You can see how to use it by looking at the `commissioner-api` README.

Visit Ngrok's website to learn more about it.

1.5 Cloze

See this article for pictures

1. Log in to you Cloze account.
2. Click on `More` at the bottom left corner of the dashboard and then click on `Settings`.
3. Under `Accounts and Services`, click on `Connect Accounts` to expand it.
4. Click on the `Add` button.
5. Select the respective service that you want to connect to Cloze.

- For Fastmail, select `Other Email` under `Other Mail and Calendar`.
- For Google account, select `Google or G Suite`.
- For Dropbox, go to `Notes and Messaging` and select `Dropbox`.

1. This will take you to the sign-in page for the respective service.
2. Sign in to the service using Nerevu credentials.

3. Review the permissions and click on `Allow`, when prompted.

1.5.1 Fastmail app password

Fastmail comes under `Other Email` type in Cloze, and therefore giving access to a 3rd-party application, like Cloze, would need an app password that will be used by that application.

1. From your Fastmail dashboard, open the dropdown menu at the top-left corner.
2. Click on `Settings` and then on `Passwords & Security`.
3. Look for `App Passwords` in this page and click on `Manage` →.
4. Enter your password at the top of this page and click on `Unlock`.
5. Click on `New App Password` button.
6. Open the dropdown adjacent to `Name`, select `Custom` and enter `Cloze` in the input box.
7. Make sure that the `Access` field is set to `Mails, Contacts & Calendars`.
8. Click on `Generate Password` and make a temporary note of it, as it will not be displayed again once you click on `Done`.
9. Use this password when you add this account in Cloze (Refer to *Linking various Accounts and Services to Cloze*).

1.6 Google Sheets API

1.6.1 Enable Google Cloud Platform

1. Go to gSuite Additional Google services
2. Search for `Google Cloud Platform` and click the checkbox
3. Click `on` in the blue bar that appears above

1.6.2 Enable Google Drive API

1. Go to Google Developers Console.
2. If you see `Google Drive API` in the list of APIs at the bottom of the screen, click it and skip the remaining steps.
3. Otherwise, click on `ENABLE APIS AND SERVICES`.
4. Search for `Google Drive API` and click on it.
5. Click on `Enable`.

1.6.3 Create API Credentials

1. Click on `CREATE CREDENTIALS`
2. Select the options as shown below:
1. Click on `What credentials do I need?`.
2. Enter the details similar to shown below (name the service account after the client):
1. Click `Continue` and save the downloaded JSON key file.

1.6.4 Access Google Sheets workbook

1. Open the Google Sheets workbook that you need to access through APIs.
2. Open the downloaded JSON key file and look for `client_email`.
3. Share the workbook with the `client_email` address and grant `Edit` permission.
4. Uncheck `Notify People` since this email address is not handled by a human.
5. Follow the respective API/library documentation for Google Sheets, for authentication and usage.

CHAPTER

TWO

RECOMMENDED WORKFLOW

2.1 Daily

2.1.1 Enter Time

- Visit the Employee Hours Google Sheet
- Find the tab titled `<YOUR NAME> (time)`
- In the `date` column, double click the first empty cell to select a date
- In the `project` column, select the appropriate project from the dropdown
- In the `task` column, select the appropriate task from the dropdown
- In the `total minutes` column, enter the number of minutes spent on the task

2.2 Weekly

2.2.1 Check hours for the week

It's expected that you already do this each work morning for the previous work day, while the details are still fresh

2.2.2 Write progress report

Details

- Due 5pm each Friday
- To: rcummings@nerevu.com
- Subject: Progress Update for week ending `MM/DD/YY`

Contents

CHAPTER

THREE

ACCOMPLISHMENTS

- Completed `foo` task
- Learned `bar` skill
- Updated documentation on `baz`

CHAPTER

FOUR

STRUGGLES / ISSUES / CONCERNS

- Couldn't figure out how to do `xyz`
- It takes too long to do `abc` on program `qrs`

CHAPTER

FIVE

FUTURE PLANS

- Complete `foo` and `bar` tasks
- Learn `baz` skill
- Updated documentation on `xyz`

CHAPTER

SIX

QUESTIONS / REQUESTS

- What is the proper way to `abc`?

CHAPTER

SEVEN

RECOMMENDATIONS

- Process `abc` is inefficient because of `xyz`. We should improve it by implementing `foo`.

CHAPTER

EIGHT

GIT AND GITHUB

8.1 Github Notifications

Add your work email address to github so that relevant notifications don't go to your personal inbox:

1. Add your work email address at the top of this page
2. Select your company email address for `nerevu` in the `Custom Routing` section at the bottom of this page
3. That's it! You rock ;)

8.2 Basic Git Flow

When working with a Nerevu repository, it is important that you create your own branch for updates, fixes, or changes. When a change needs to be implemented, you simply commit and push your changes to your own branch and then submit a pull request to request merging your branch with origin. This prevents your changes from adversely affecting production code.

1. Clone repository

```
git clone https://github.com/nerevu/<REPO_NAME>.git
```

1. Create your branch to add your fix/feature

```
git checkout -b <BRANCH_NAME>
```

1. Prettify your code

Python

```
manage prettify
```

JavaScript

```
npm run prettify
```

1. Push your commits to Github and create a Pull Request

```
git commit -m "<COMMIT_MESSAGE>"
git push --set-upstream origin <BRANCH_NAME>
```

NOTE: As a general rule, don't merge code that purposefully raises errors and kills the application. Talk to Reuben about how to gracefully handle situations where you want to throw an error.

If master changes while you are fixing a pull request, rebase to master (please ask questions if you do not feel confident rebasing)

```
git fetch origin
git rebase origin/master
```

Fix conflicts and force push to YOUR BRANCH on Github.

```
git push -f origin <BRANCH_NAME>
```

Never force push anything to `MASTER`! If you feel you must, speak with Reuben in great detail first!

8.3 Project Management

8.3.1 Work Backlog

You can find issues that are being worked at the Nerevu Group Github Projects page. If you click on the project you are working on, you will find a Kanban Board with all the issues (some of them have milestones with due dates).

8.3.2 Kanban Board

When you start working on a new issue, ensure you immediately do the following to keep the Kanban Board up to date:

1. Create a Pull Request (PR) and select what `Project` it belongs to. This automatically adds your PR to the `In Progress` column on the Kanban Board. The `Assignees`, `Labels`, and `Milestone` sections are only needed for issues (the issue needs `Project` as well.
2. Go to the Kanban Board for the respective project and move the related issue from the `To Do` column to the `In Progress` column.
3. Once done fixing the issue, rebase to squash the work into one commit `git rebase -i` and add the words `(closes <ISSUE_NUMBER>)` to the commit message. This will automatically close the corresponding issue once your PR gets merged.
4. When your PR is merged into master, both the Issue and the PR in the Kanban Board will automatically move to the `Done` column.

8.4 Misc

8.4.1 Commit Messages

Your commit message should adhere to the following:

- be no more than 50 characters
- give a descriptive summary of the work accomplished
- begin with a present tense verb
- use sentence case

Examples:

```
Remove unused attributes
Refactor widgets
Update models and db initialization
```

If your commit fits one of the following categories, add the appropriate prefix.

- `[FIX]`: Fixes a bug
- `[CHANGE]`: Changes existing behavior or external API
- `[ENH]`: Enhances existing behavior
- `[NEW]`: Adds new behavior
- `[CVE]`: Patches a security vulnerability

Examples:

```
[FIX] Set default cache timeout
[CHANGE] Update college data model
[ENH] Optimize data fetching
[NEW] Make number of months configurable
[CVE] Patches elliptic CVE-2020-13822
```

If your commit fixes an existing issue, add the suffix `(closes <ISSUE_NUMBER>)`.

Example:

```
[ENH] Standardize number format (closes #10)
```

8.4.2 Commit Frequency

You should be committing your code and pushing to GitHub often so that your changes are always available for others to see. Even if the commits don't have completed code, still commit them. You should strive to have GitHub up-to-date with your local branch at all times. Once you are ready to merge your code into the master branch, rebase your commits into more logical groups. For example:

1. All commits that fixed an issue should be grouped into one commit.
2. Other code changes that are not related to an issue should be grouped into their specific commit tasks.

8.4.3 When to Rebase

There are several times that you should consider rebasing in Git.

1. Periodically to the master branch to make sure your code doesn't break the new changes to the master branch.
2. When you make a change that a reviewer requested in a PR.
3. Any other time that makes sense (you'll learn this over time)

8.4.4 How to Rebase

Rebasing is necessary whenever history needs to be changed. This can be for changing commit messages or for changing the order and contents of commits. For example, if you want to change commit messages follow these steps:

1. Run `git rebase -i master`
2. Change the `pick` field to `edit` next to the commit you intend to change
3. Save the file
4. Run `git rebase --continue`
5. This will reopen the rebase file where you can change the commit text and exit to save

Similarly, you can edit the contents of the files themselves when under edit mode. After step 2 above, the local file system will reflec the changes of the selected commit. Here you can change the files using a text editor and then recommit them like so:

1. Run `git rebase -i master`
2. Change the pick field to `edit`
3. Make changes to the file system
4. Run `git status` to see edited files
5. Run `git add .` followed by `git commit --ammend`
6. Complete the rebase using `git rebase --continue`

This will effectively change the edits included in that specific commit.

On the other hand, if you want to erase a commit entirely you can simply comment out the commits you wish to erase using # and then continue the rebase. This will edit the commit history to erase any commits you erase.

8.4.5 .gitignore File

- don't commit `cache` folders and files (add them to `.gitignore`)

CHAPTER

NINE

NEREVU CODING STYLE

The following examples will help you understand how you will be expected to code at Nerevu. Read through them and ask questions if you don't understand.

9.1 Python/Flask

- Avoid complexity in your code. If it's hard to understand what's going on in a function, break it out. It shouldn't take a lot of explaining to show what code is doing. Commenting will help with this.
- Most Nerevu projects have a `manage.py` file that has CLI commands that can be run for a project using the following CLI call - `manage {function_name}`. If the project you are working on has a `manage.py` file with a`prettify` function in it, you should run `manage prettify` on your code before your PR is merged. This keeps the formatting consistent.
- Practice `DRY` coding (Don't Repeat Yourself). If you find you are writing similar code over and over, try abstracting it out into a function (if it makes sense to do so). The below example is trivial, but should get the point across.

```
# INCORRECT
user1 = 'doug peterson'
user1 = " ".join(name.capitalize() for name in user1.split(" "))
user2 = 'george clooney'
user2 = " ".join(name.capitalize() for name in user2.split(" "))

# CORRECT
def format_name(name):
    return " ".join(name.capitalize() for name in user1.split(" "))

user1 = 'doug peterson'
user1 = format_name(user1)
user2 = 'george clooney'
user2 = format_name(user2)
```

- Start thinking like a functional programmer. Don't mutate objects. Return new objects instead. See the docs for more info on functional programming in python.

```
users = [{"name": "brian"}, {"name": "jenna"}]

# INCORRECT
def lowercase_name(user):
    user["name"] = user["name"].lower()
    return user
```

(continues on next page)

(continued from previous page)

```
list_of_users = map(lowercase_name, users)

# CORRECT
def lowercase_name(user):
    return {**user, "name": user["name"].lower()}

list_of_users = map(lowercase_name, users)
```

- use `pos` instead of `i` when using enumerate (with `range()` you don't need to use this)

```
names = ['bob', 'jack', 'bill', 'jessica']

# INCORRECT
for i, name in enumerate(names):
    print(f"{i}: {name}")

# CORRECT
for pos, name in enumerate(names):
    print(f"{pos}: {name}")
```

- Generators

Name Generators `gen_{function_name}` Use generators instead of appending to empty lists

```
# INCORRECT
letters = []

for i in range(10):
    if i % 2:
        letters.append(chr(i))

# CORRECT - notice the naming of the function as well as the (gen_)

def gen_letters(nums):
    for i in nums:
        if i % 2:
            yield chr(i)

letters = list(gen_letters(range(10)))
```

- use list comprehensions instead of list and map

```
names = ['Phillip', 'Annette']
lowercase_names = lambda n: n.lower()

# INCORRECT
new_names = list(map(lowercase_names, names))

# CORRECT
new_names = [lowercase_names(n) for n in names]
```

- use `dict.items()` instead of `key in dict`

```
obj = {"name": "brian"}

# INCORRECT
for key in obj:
```

(continues on next page)

(continued from previous page)

```
    print(obj[key])

# CORRECT
for key, val in obj.items()
    print(val)
```

- make variable names meaningful unless they are in a list comprehension or lambda function, etc..

```
users = [{"name": "brian"}, {"name": "jenna"}]

# INCORRECT
def gen_users(n):
    yield n['name']

list_of_users = list(gen_users(users))

# CORRECT
def gen_users(user):
    yield user['name']

list_of_users = list(gen_users(users))

# ALSO CORRECT
list_of_users = [u["name"] for u in users]
list_of_users = map(lambda u: u["name"], users)
```

- If you're not sure if a key will be present in a dictionary, use `dict.get("key")` so your code doesn't break.

```
user = {"name": "brian"}

# INCORRECT
age = user["age"] # Throws a KeyError

# CORRECT
age = user.get("age") # returns None
```

- Add breaks to short circuit for loops once the correct field is found

```
numbers = [1,2,3,4,5,6,7,8,9,10]

# CORRECT EXAMPLE
contains_number_four = False

for number in numbers:
    if number is 4:
        contains_number_four = True
        break # prevent the loop from continuing!
```

- Nested loops in general are code smell really - `Set` datatypes are indexed, so doing `if x in Set` is faster than doing a double for loop

```
names = ["Bob", "Jesse", "Alyssa", "Frank"]
whitelist = {"Bill", "Jack", "Frank", "Frank"}

# These functions yield names that are present in a whitelist
```

(continues on next page)

(continued from previous page)

```
# INCORRECT
def gen_names():
    for name in names:
        for other_name in whitelist:
            if name == other_name:
                yield name

# CORRECT
def gen_names():
    for name in names:
        if name in whitelist:
            yield name

# MORE CORRECT
def gen_names():
    for name in set(names).intersection(whitelist):
        yield name
```

- Don't use `types` (e.g. str, list, dict, etc.) in your variable names

```
# INCORRECT VARIABLE NAMES
numbers_array = [1,2,3,4,5,6,7,8,9,10]
user_dict = {'name': 'bob', 'age': 16}
names_str_list = ['bill', 'george', 'katie', 'geoffrey', 'jessica']

# CORRECT VARIABLE NAMES
numbers = [1,2,3,4,5,6,7,8,9,10]
user = {'name': 'bob', 'age': 16}
names = ['bill', 'george', 'katie', 'geoffrey', 'jessica']
```

- Don't commit commented out code.

```
name = 'bob'

# DON'T COMMIT THIS
# name += "cat"
# print(name)
```

- Most Nerevu projects have a `manage.py` file that has CLI commands that can be run for a project using the following CLI call - `manage {function_name}`. If the project you are working on has a `manage.py` file with a`prettify` function in it, you should run `manage prettify` on your code before your PR is merged. This keeps the formatting consistent.

9.2 Coffeescript/Mithril

- prefer `is` and `isnt` over == and `!=`
- prefer `unless ctrl.page()` to `if ctrl.page() is undefined`
- coffeescript vars should be pascalCase
- Loop through objects like this: `Object.entries(statsByRep).map ([rep, repStats]) =>`

CHAPTER

TEN

NEW PROJECTS

10.1 Create a new Project

Start new projects by following the steps in the README of a Nerevu Cookiecutter repo.

10.2 ENV file

A file for your environment variables.

1. Create an `env` file in your Dropbox folder

```
touch Dropbox/Nerevu/Security/<YOUR_USERNAME>/<PROJECT_NAME>-env
```

1. Navigate to your project directory

```
cd /PATH/TO/<PROJECT_NAME>
```

1. Create a symbolic link from the env file to your new project

```
ln -s Dropbox/Nerevu/Security/<YOUR_USERNAME>/<PROJECT_NAME>-env .env
```

10.2.1 Example

Let's say you create a new project called CashFlowAPI and your username is `rcummings`. You should create an env file named `cashflow-api-env` in the `Nerevu Group Dropbox/Security/rcummings/` directory. You should then run `ln -s Nerevu Group Dropbox/Security/rcummings/cashflowapi_env .env` from the root folder of your new project (see this page to learn more about symbolic links).

We use python-dotenv to manage environment variables. Make sure you install this package into your new project so your environment variables get picked up by your application.

10.3 Python Requirements

You should set your python projects up with 3 different requirements files that contain references to the packages needed by the program.

1. `base-requirements.txt`
    - Contains the necessary packages for most Nerevu APIs to run (you'll find examples of this in other repositories on GitHub).
2. `requirements.txt`
    - Contains an import statement for the `base-requirements.txt` file.
    - Contains other necessary packages for the current project to run.
3. `dev-requirements.txt`
    - Contains necessary and/or useful packages for developers working on the current project.

CHAPTER

ELEVEN

RESOURCES

11.1 Recording Time

- Don't dwell on something too long. Just move to a different task.

11.1.1 Timely

- Getting Started with Timely

Timely uses the 2 general concepts to organize completed tasks: `Projects` and `Tags`. `Projects` can be generally grouped into two categories: `Internal` and `Client`.

- `Internal` projects are assigned to the `Nerevu` client and is work that is not tied to a direct client, but is necessary to company operations (checking mail, meetings, general learning, etc.)
- `Client` projects are assigned to an individual client, and is work that is directly related to a client project (developing features, reviewing code, etc.).

Similarly, `Tags` come in two categories: `Billable` and `Non-Billable`....

When inputting hours worked, tags should be applied to every task we do, whether internal or external, as well as a simple description of the work we did.

NOTE: When using the `learning` tag, the work is general in nature unrelated to a specific project (e.g., functional programming), please use the `Internal` project. However, if the learning is in relation to a specific project you are working on (e.g., Flask Admin), please use that client project instead of `Internal`.

11.2 ERP

11.2.1 Dropbox

- Nerevu Group Team Space
- Getting Started with Paper

11.2.2 Cloze

- Getting Started with Cloze
- Where can I find my Cloze API key?
- API documentation
- Add GSuite and Dropbox Accounts

11.2.3 Fastmail

- Getting Started with Fastmail
- Sharing calendars
- Syncing calendars
- Device setup

11.3 Development

11.3.1 Readings

- The Hitchhiker's Guide to Python!
- pre-commit hooks
- Python Functional Programming Modules

11.3.2 Talks

- Structure and Interpretation of Computer Programs
- Raymond Hettinger - Beyond PEP 8
- Python Concurrency From the Ground Up: LIVE!
- Simple Made Easy

11.4 Design

- logos (color, monochrome)
- fonts

11.5 Devops

- Continuous Integration

CHAPTER

TWELVE

CKAN

12.1 On AWS EC2

- Upgrading CKAN
- Configuring CKAN with https
- Making CKAN live

CHAPTER

THIRTEEN

MISC

13.1 Common Issues

13.1.1 Forgot the Environment File - API Connection Issues

Every project should have a `.env` file associated with it. You can find these files stored as described here. If you don't have a `.env` file, you will get confusing errors when trying to connect to an API because you will be missing your `client_id` or other important information. To get the `.env` file in your project, check the `Nerevu Group Dropbox` for one and create a symbolic link to it. If you don't find one in Dropbox, create one and add it to Dropbox.

13.2 Retainer Invoicing

- Create a non project linked invoice billed to "Retainer Fees" (example)
- Create a project linked credit note paid from "Retainer Fees" (example)
- Once paid, reconcile the bank transfer
- Apply credits to future retainer based work

CHAPTER

FOURTEEN

YOUR FIRST WEEK AT NEREVU

Ensure that you complete all of the following

14.1 Access your new email account

Login to Fastmail with the following information:

- username: <YOUR_FIRST_INITIAL><YOUR_LAST_NAME>@nerevu.com, e.g., rcummings@nerevu.com
- password: separately emailed to you

14.2 Change your password

Instructions

14.3 Complete your Gusto profile

Follow the instructions contained in the email with the subject *Let's get you set up with Nerevu*.

14.4 Complete Paperwork

- Sign and return your Employment Agreement
- Schedule a time for your orientation and ensure you have your passport (or drivers license and birth certificate) present for the meeting
- Add a headshot image and a 1 paragraph biography to Dropbox

14.5 Schedule Weekly Review

- Schedule a time for your reoccurring weekly Monday review meeting

14.6 AWS Help

By default, Nerevu Group's EC2 instances are in the N. Virginia region. To get access to EC2 instances via SSH, go here.