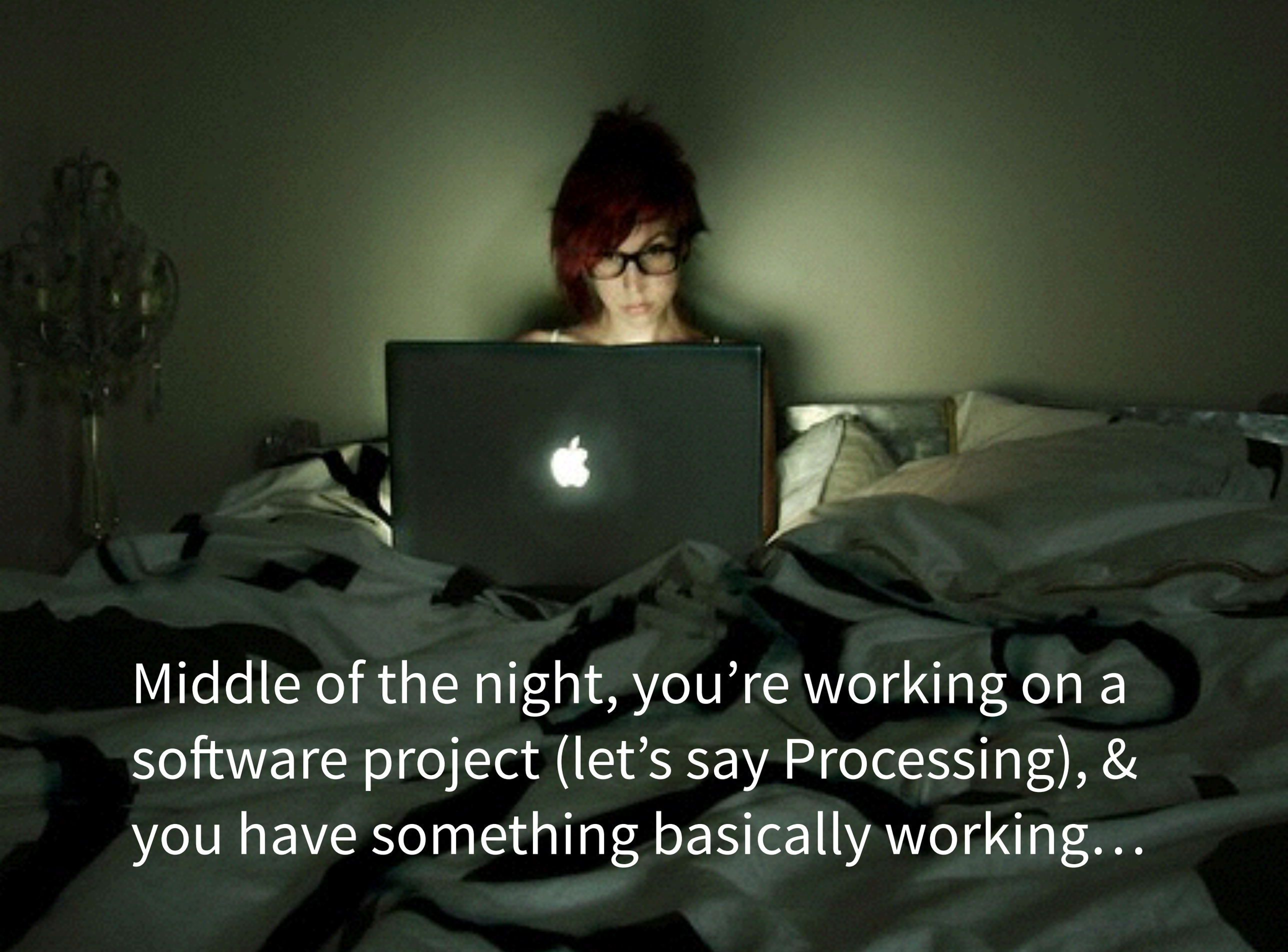
The Basics Every Coder Needs to Know

Dan Wilcox
Visiting Teaching Assistant Professor
Emergent Digital Practices

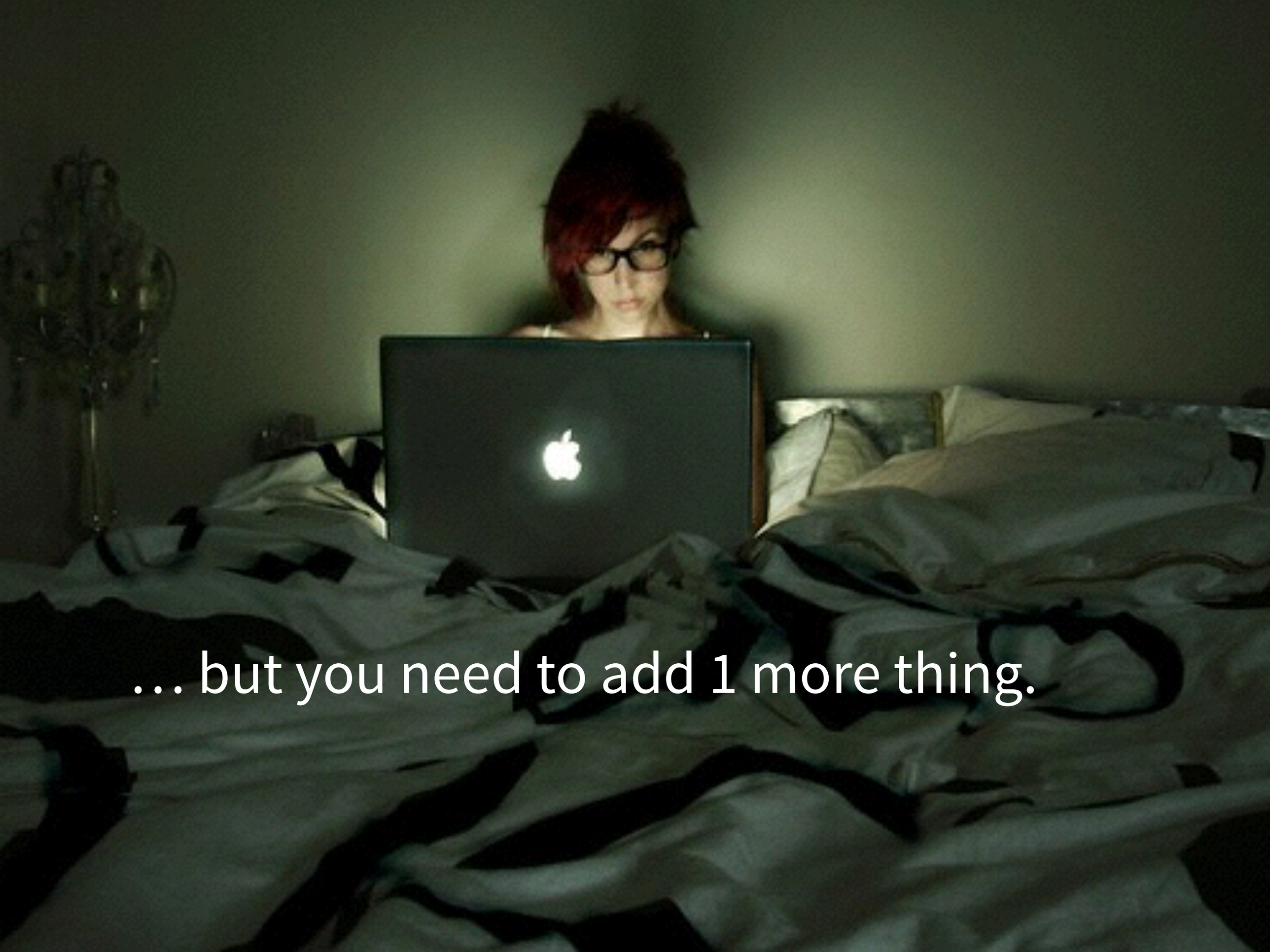
Winter 2016

class.danomatika.com

Hypothetical Situation 1



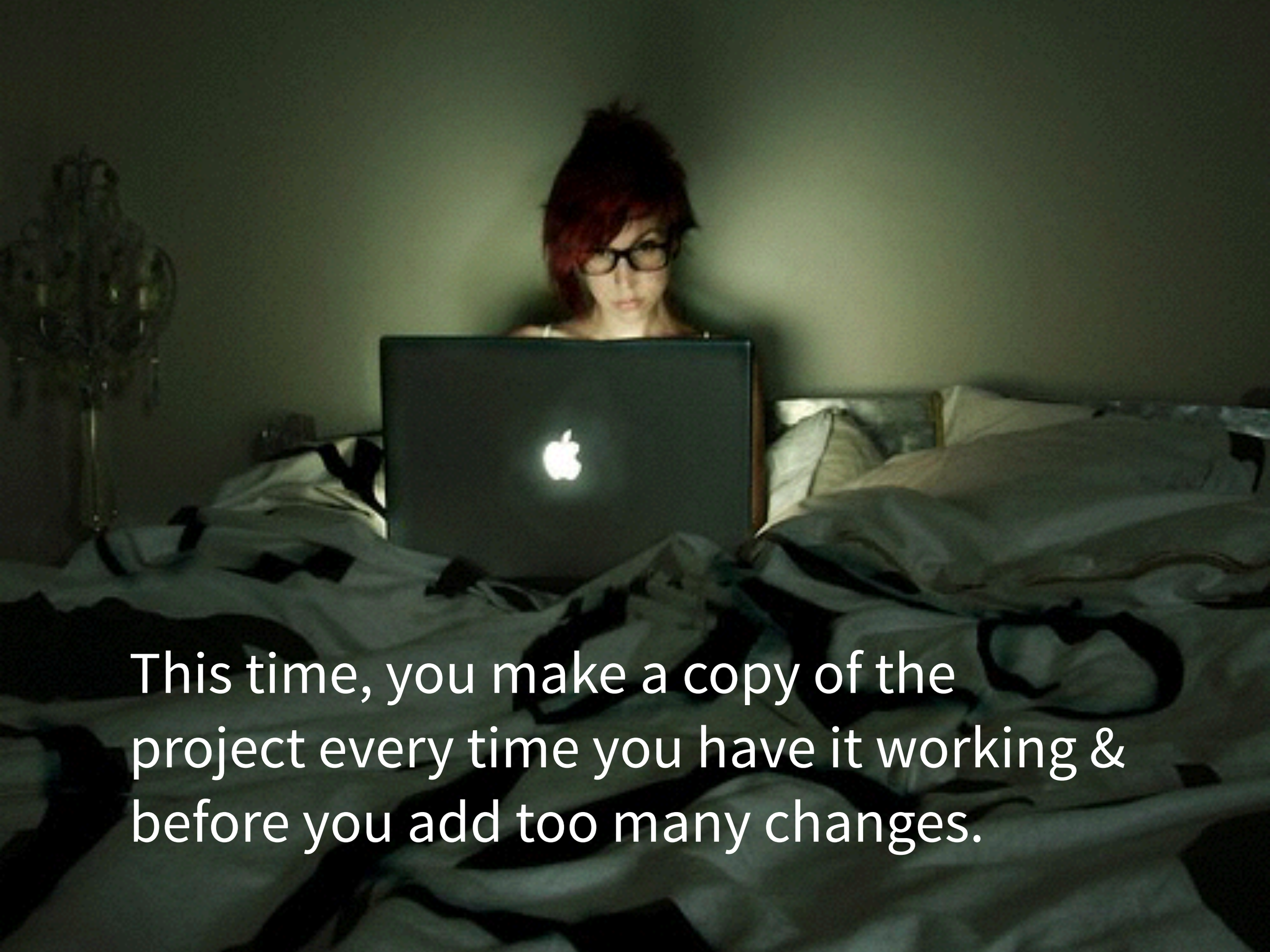
Middle of the night, you’re working on a software project (let’s say Processing), & you have something basically working…


. . . but you need to add 1 more thing.

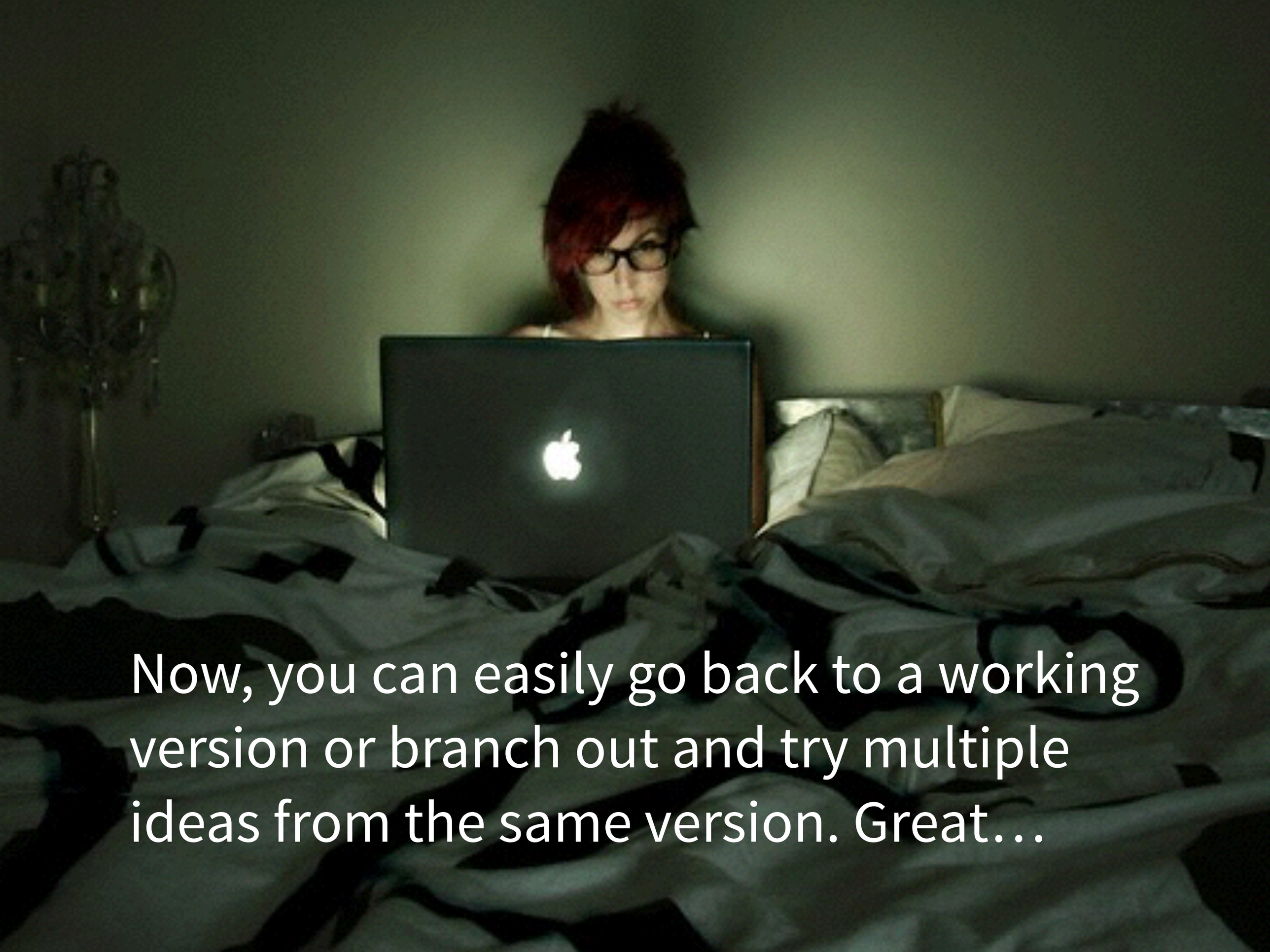
Everything is broken & you don’t
remember what you changed.

It’s going to take **forever** to get it working again.

Hypothetical Situation 2



This time, you make a copy of the project every time you have it working & before you add too many changes.



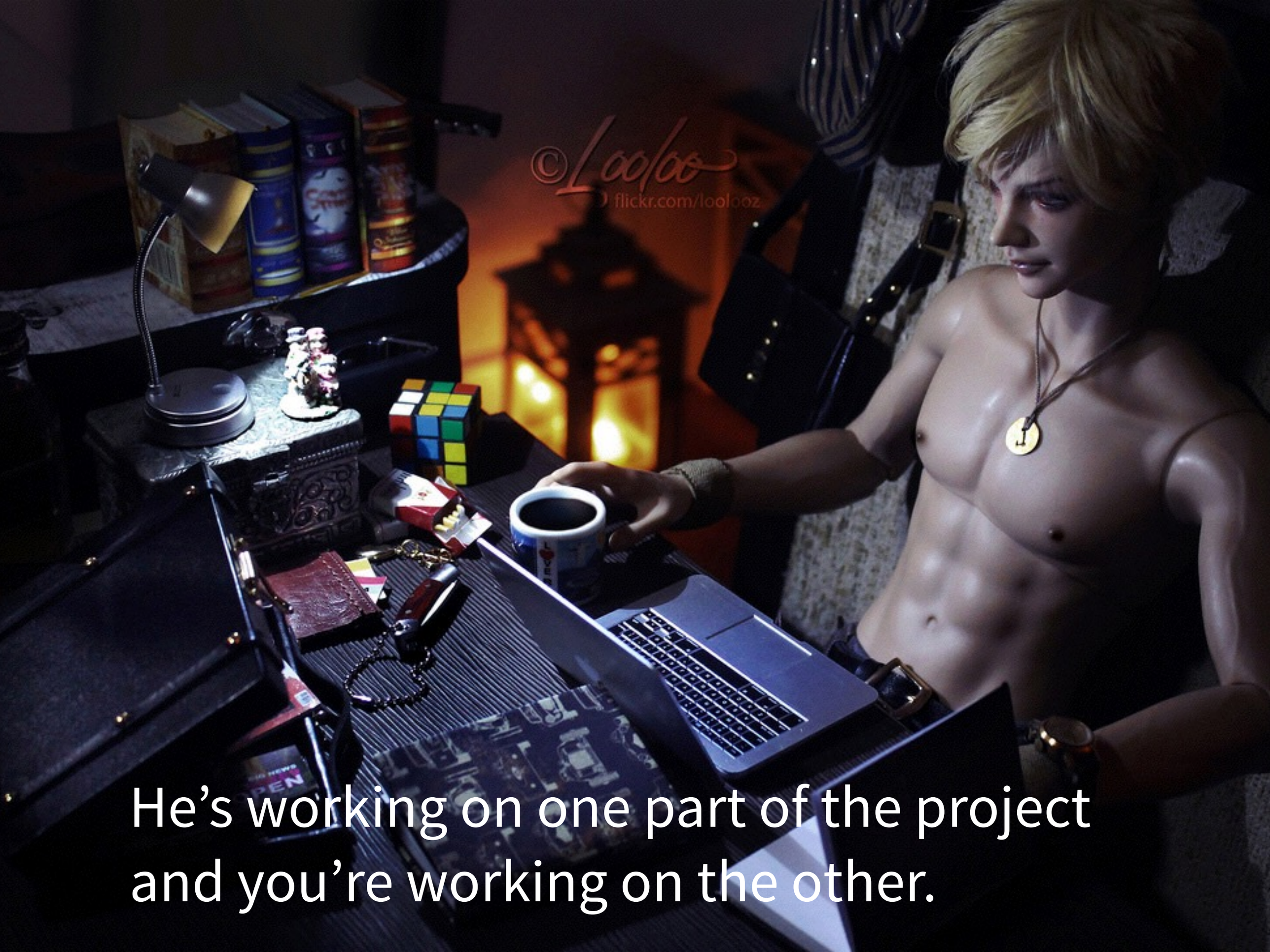
Now, you can easily go back to a working version or branch out and try multiple ideas from the same version. Great…

… except now you start to have **lots** of copies of the same project.

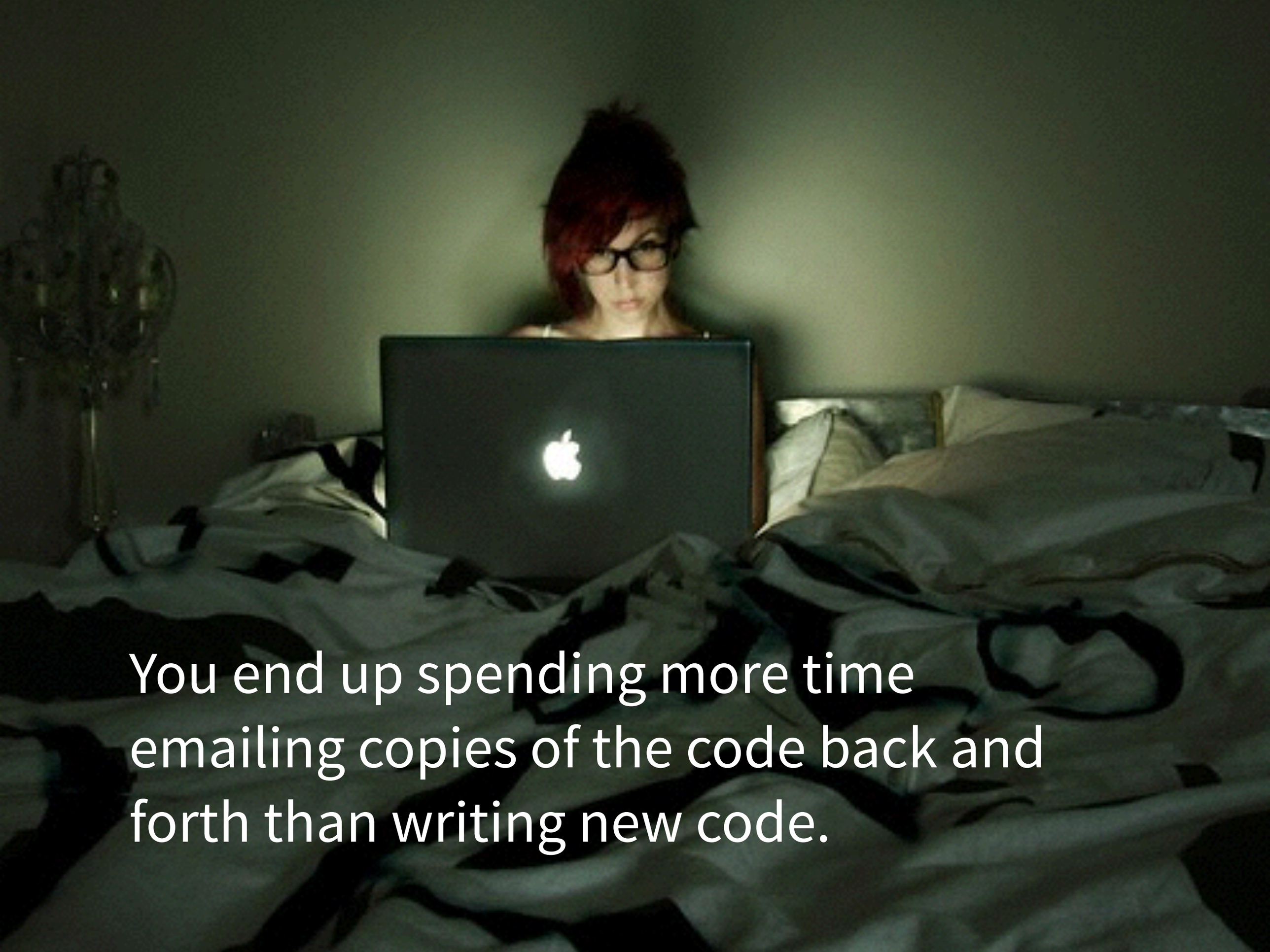
And combining code between different versions is a **huge** pain.

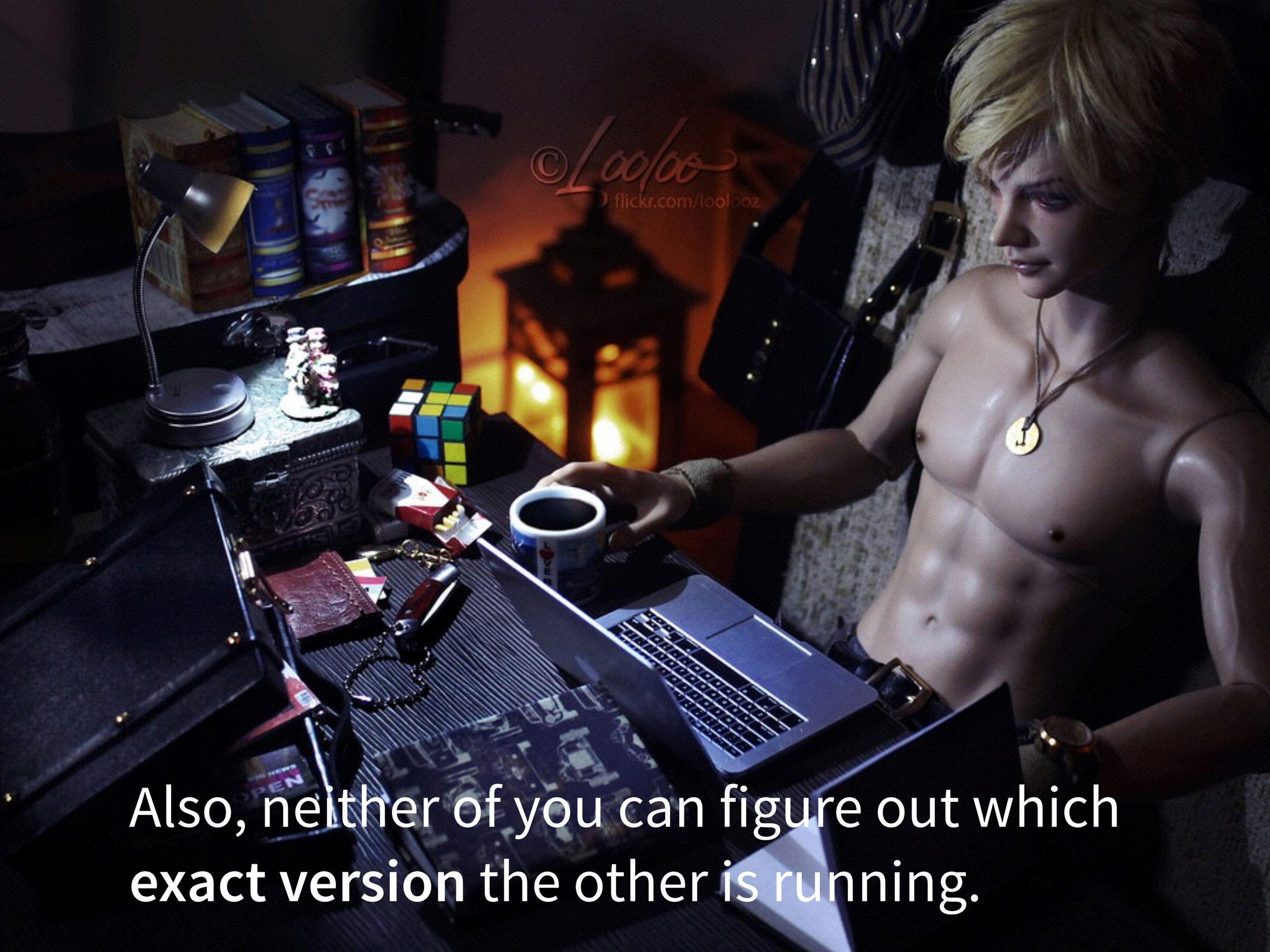
Hypothetical Situation 3

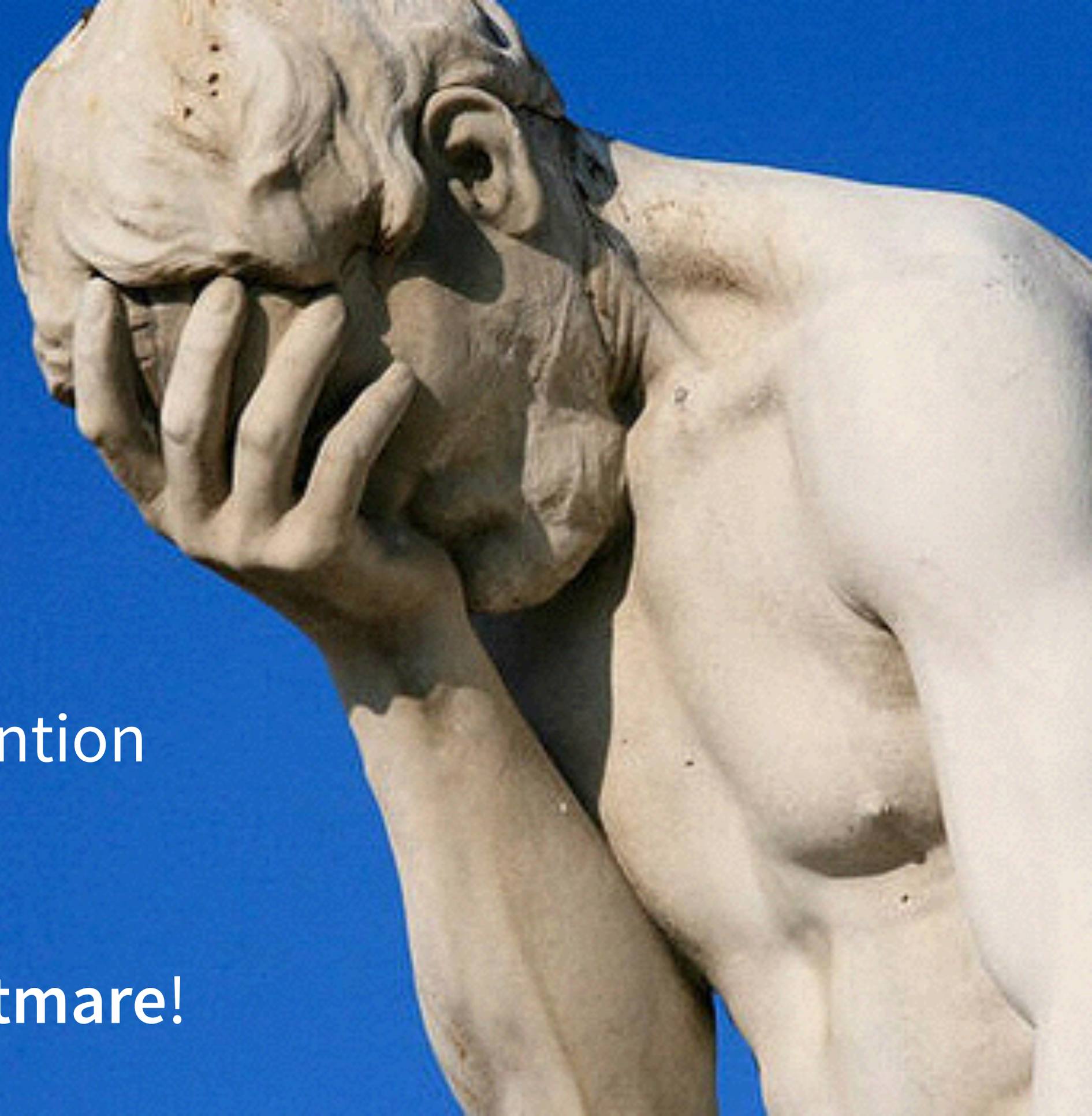
You’re working on a collaborative project with a friend.



He’s working on one part of the project and you’re working on the other.


You end up spending more time
emailing copies of the code back and
forth than writing new code.


©Looloo
flickr.com/loolooz
Also, neither of you can figure out which
exact version the other is running.



Not to mention
conflicts.

It’s a **nightmare**!

There **has** to be a better way!


VERSION CONTROL
ALL THE THINGS
memegenerator.net

Let the computer keep track of changes and versions.

This is called *source control.*

There are a number of tools out there which do this, one of them is…

git

Open Source

Distributed Source Control Management

Originally written for the Linux Kernel

Command line tool & graphical front ends (gitk, Github Desktop, etc)

Source Control

aka

Keeping track of
changes & versions

Optimized for source files (text)

Not good for large binary files like video and high resolution images

Has a funny name (kind of like “get”, British slang, etc)

git

This is probably your current workflow…

Basic Git workflow:

- make changes
- ***add*** new and/or modified files
- ***remove*** files you don't want
- ***commit*** changes (aka save)

Work in your project folder and git remembers the changes for you.

A git-managed project is a ***repository.***

"***repo***" for short

This is just a fancy name for a folder you told git to keep track of.

git only keeps track of things you add

Changes can be seen via highlighting differences been old & new versions:

```
- this was the original text on this line
+ it has now been replaced by this
```

git keeps track of these differences and can undo/redo them for you.

Each commit has a unique identifier and you can return one at any time.

It’s like a “save point” & you can add a message reminding you what you did:

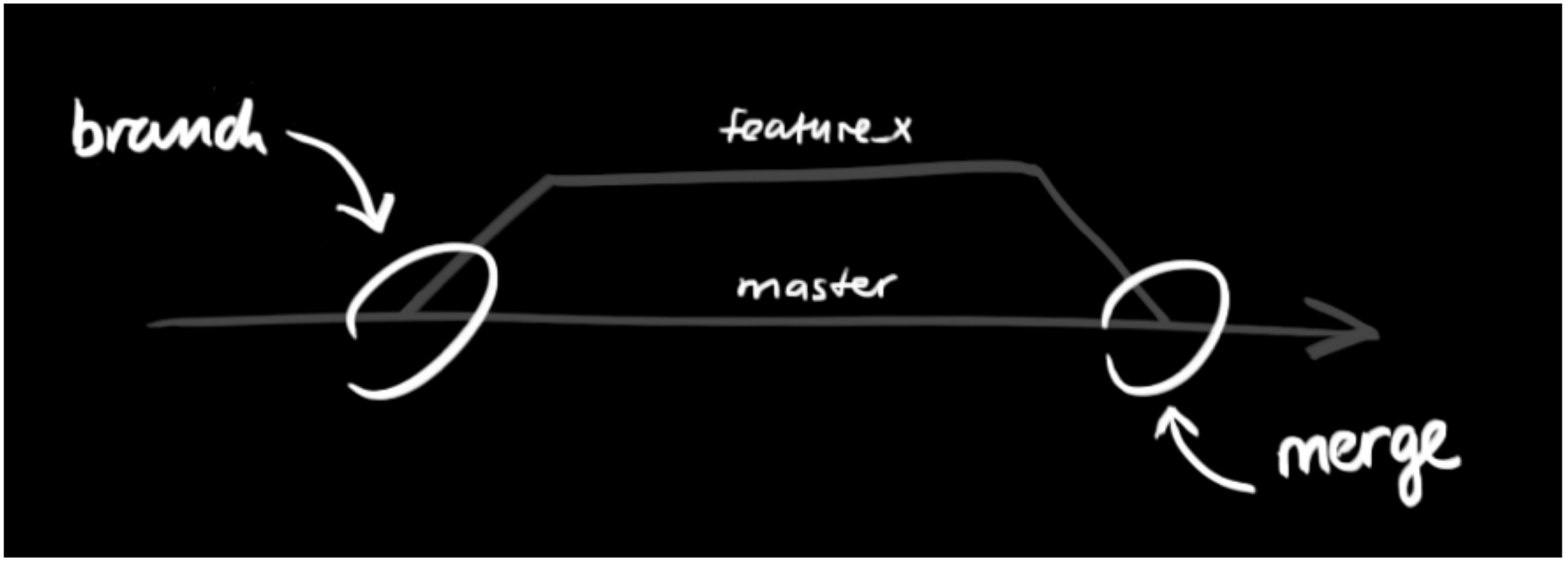
```
commit 84d743774e039e8197f7f10e467857763eb4494a
Author: danomatika <danomatika@gmail.com>
Date:   Mon Jan 18 12:27:37 2016 -0700

    fixed bug a, fixed bug b, & added feature 4 (woohaa)
```

You can also name specific commits so it’s easier to find them again.

This is called ***tagging*** the commit.

Software projects generally tag versions aka 0.2.0, 0.3.1, 1.0.2, etc but you can name them whatever you want.

You can isolate ideas & work from the main “working code” using ***branching***.



git

Every git repo has a ***master branch*** by default and you can create a new branch off of any commit at any time.

You can then make multiple commits in the branch until you finish something, then ***merge*** the changes back into the ***master branch.***


LETS TAKE THE CONTENT
FROM THIS BRANCH
AND PUT IT INTO THIS
BRANCH
quickmeme.com

This means you can easily try out ideas without hosing your ***master branch.***

aka

CREATIVE FREEDOM

git

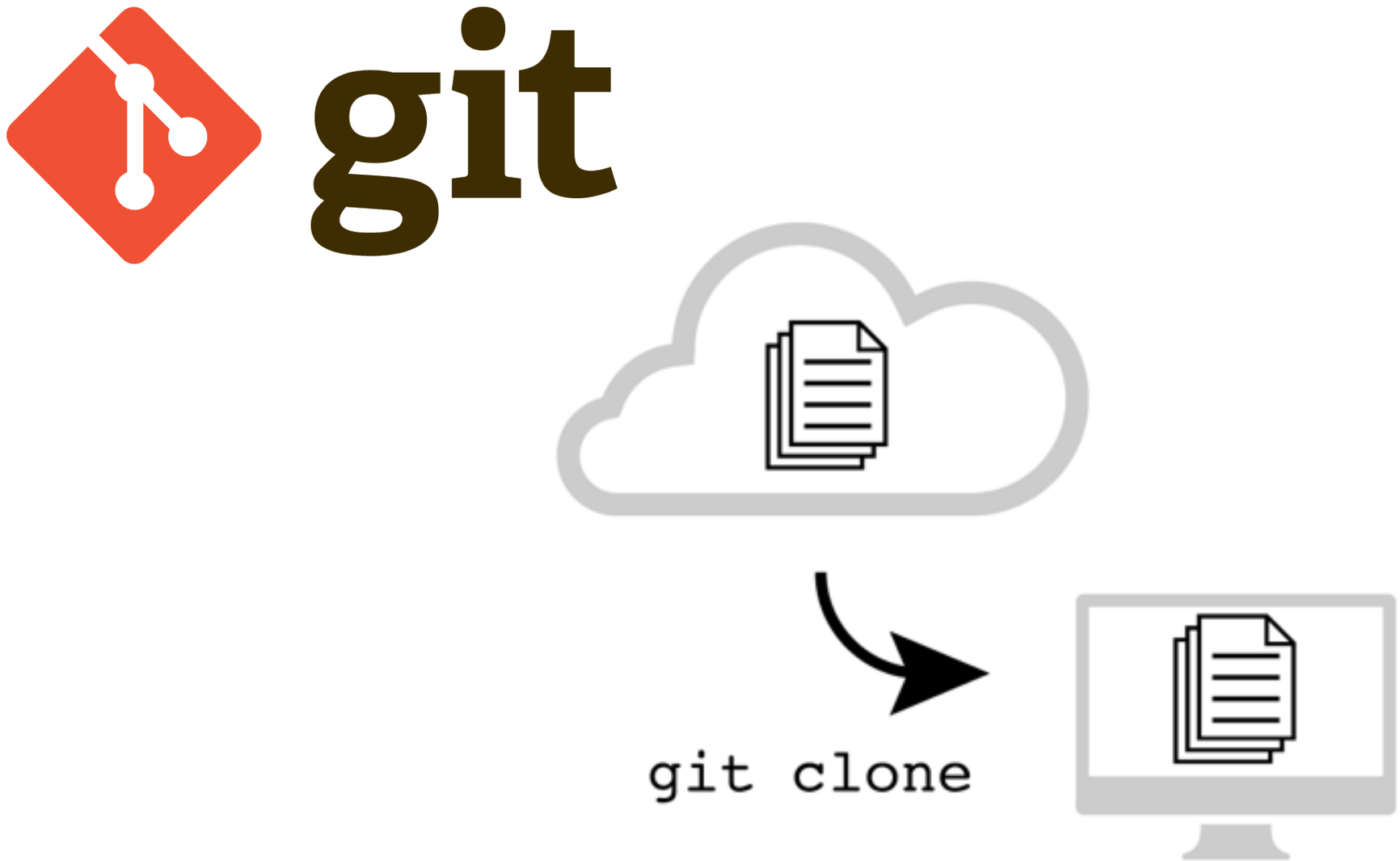
Distributed: not centralized

Work locally on your computer, then…

push changes to *another location*

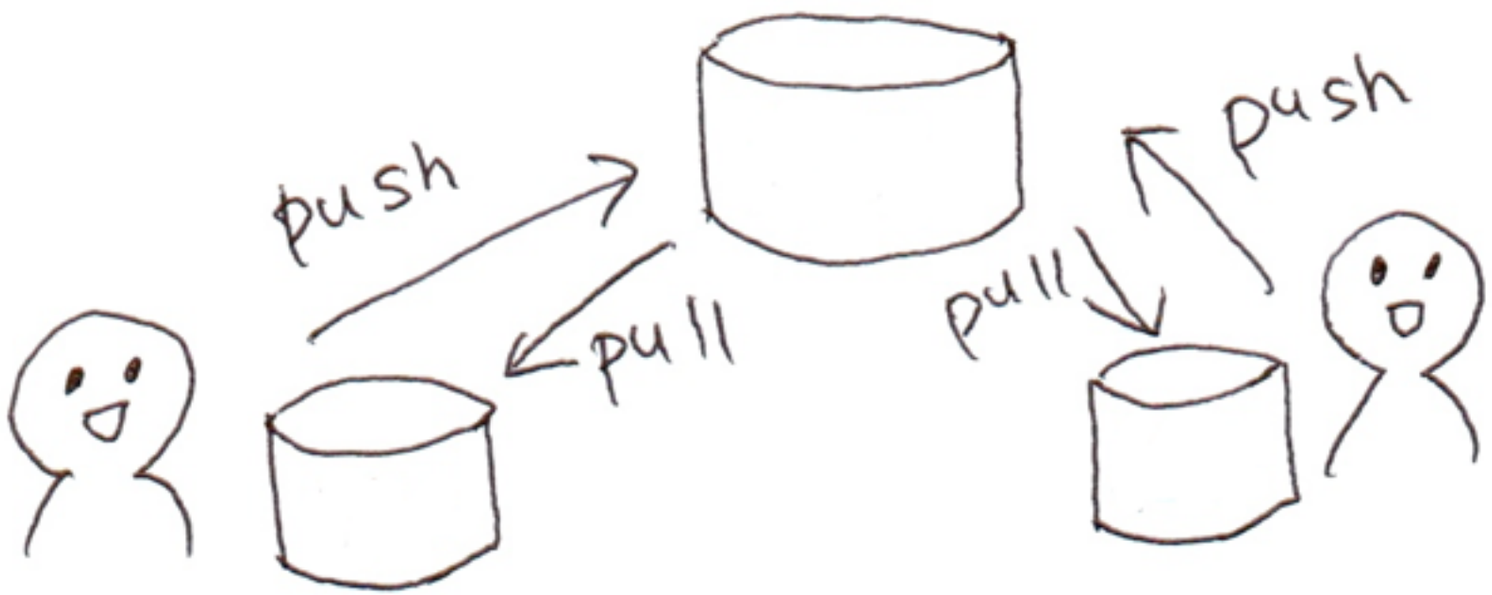
or

pull changes from *another location*



You can get a copy of a remote repo if you know it's location or web address.

This is called ***cloning*** a repo.



Once cloned, you then ***pull*** in any new changes and ***push*** your changes.
(If you have permission, that is!)

This allows multiple people to share code through remote locations…

A *remote location* can be:

- another folder
- another computer
- a web server
- or an online service…

GitHub

git

github
SOCIAL CODING

GitHub **uses** git, but did **not** create it
(*no matter what the hype says*)

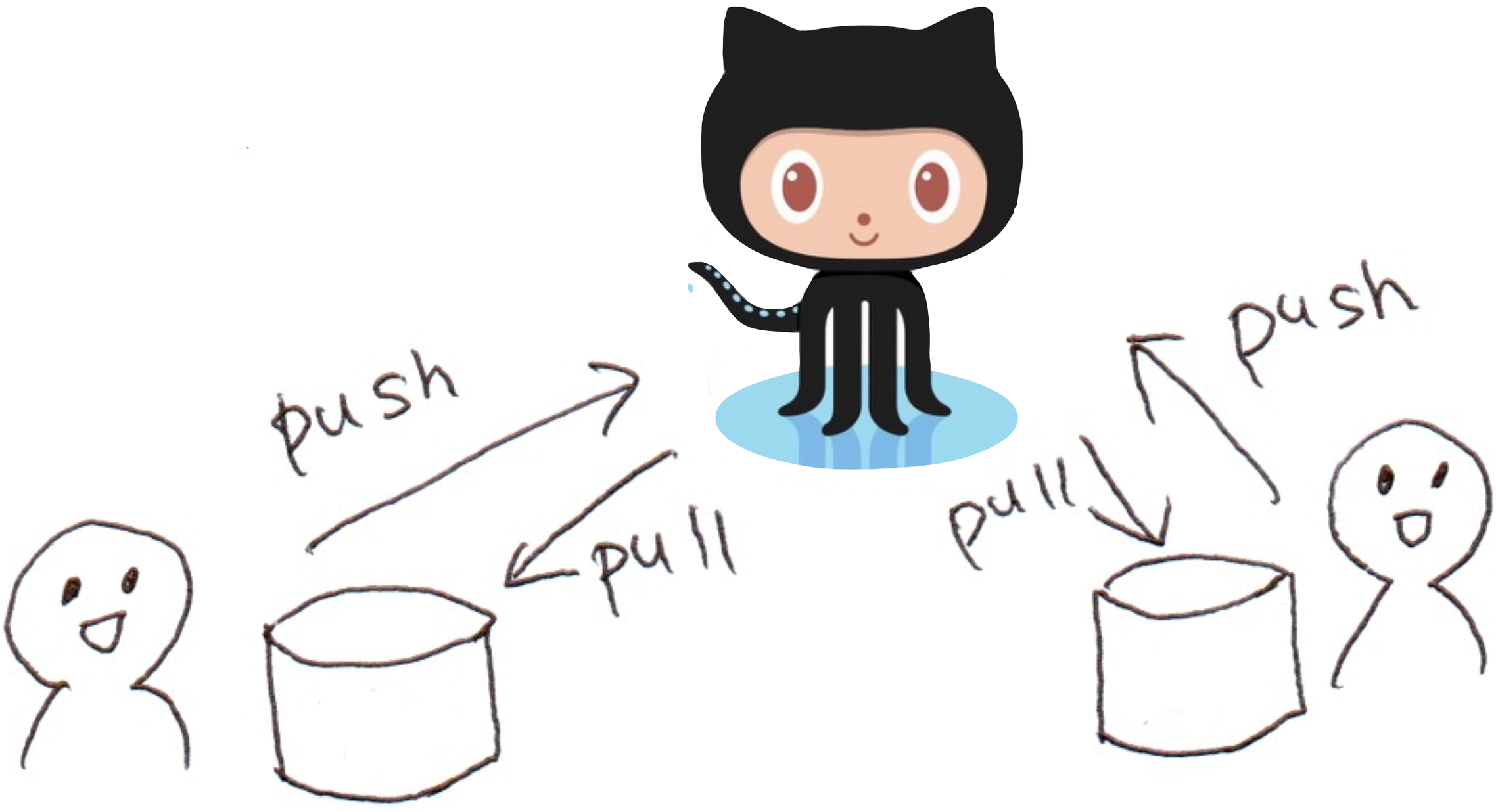
You can also use git **without** GitHub

GitHub

GitHub is basically a web service which provides a place to store git repositories.

It has a number of social features which make it easy to collaborate on a project with other people, many of whom you may not ever meet in person.

GitHub



GitHub

You can set up an account for free and create an unlimited number of *public* repositories at no charge.

Private repos, on the other hand, will cost you money.

Most repos are *public*.

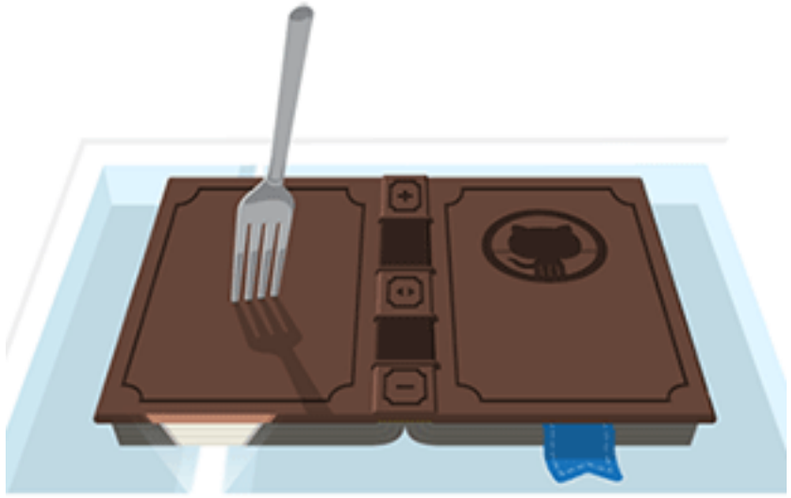
GitHub

Public repo example: Processing

Features:

- source code
- issues (aka bugs, etc)
- wiki
- commit, tag, & branch views

GitHub



Most useful feature: the ***pull request***

Basically, you can make your own copy of a public repo on GitHub into your user account. This is called a ***fork***.

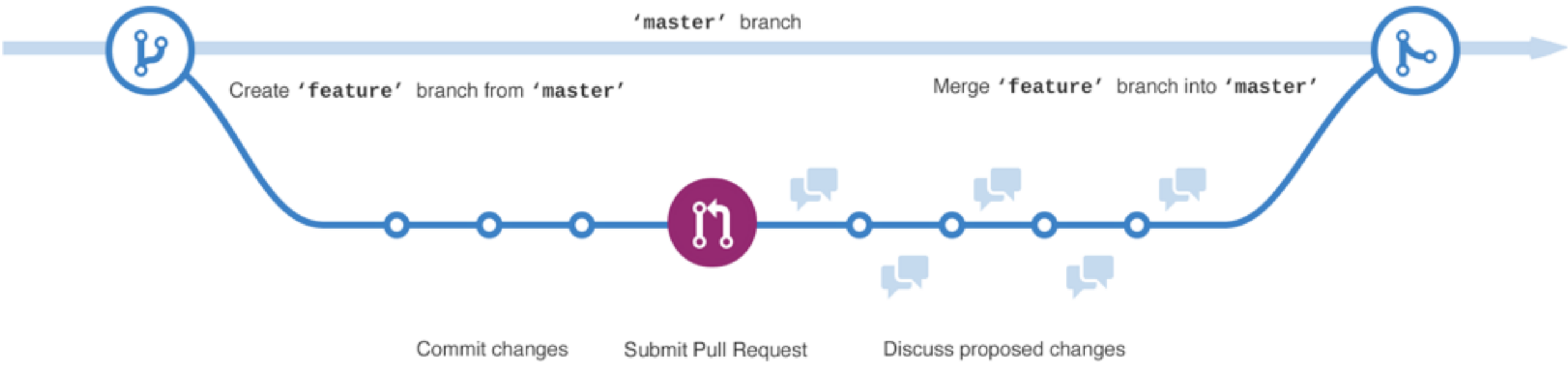
Do you own work, then submit it back to original repo via a ***pull request***.

GitHub

Do you own work, then submit it back to original repo via a ***pull request***.

The ***pull request*** allows people outside of the main repository developers to submit changes.

GitHub



Thus, the *PR* enables social collaboration without the direct need for permission or experience.

GitHub

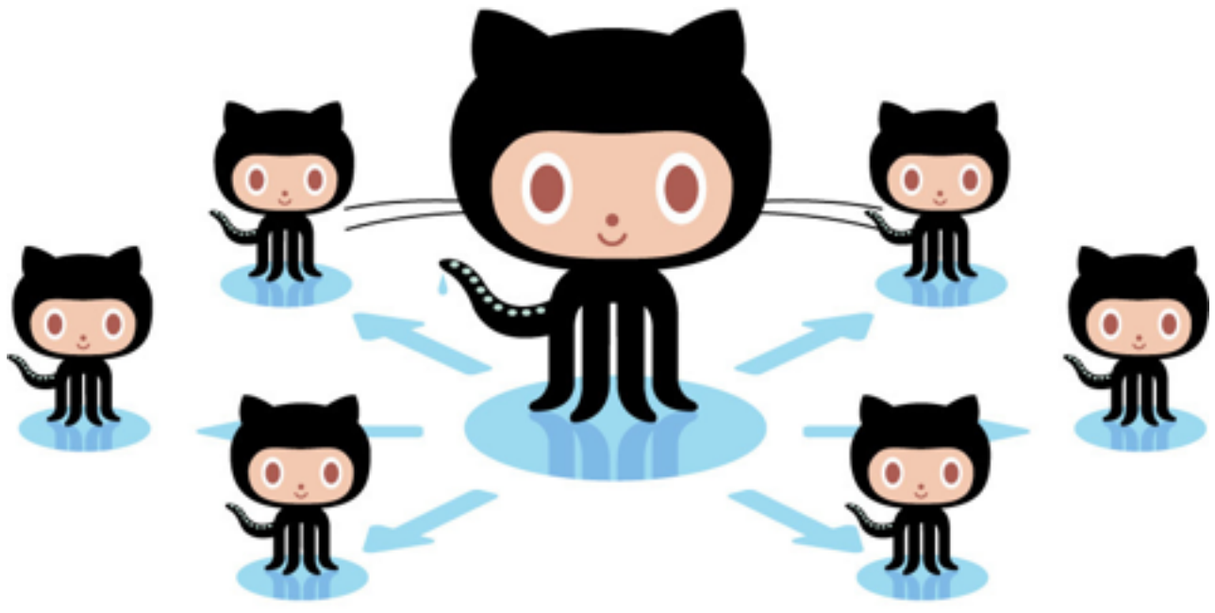
Anyone can help open source projects.

For instance, found a typo in the Processing docs?

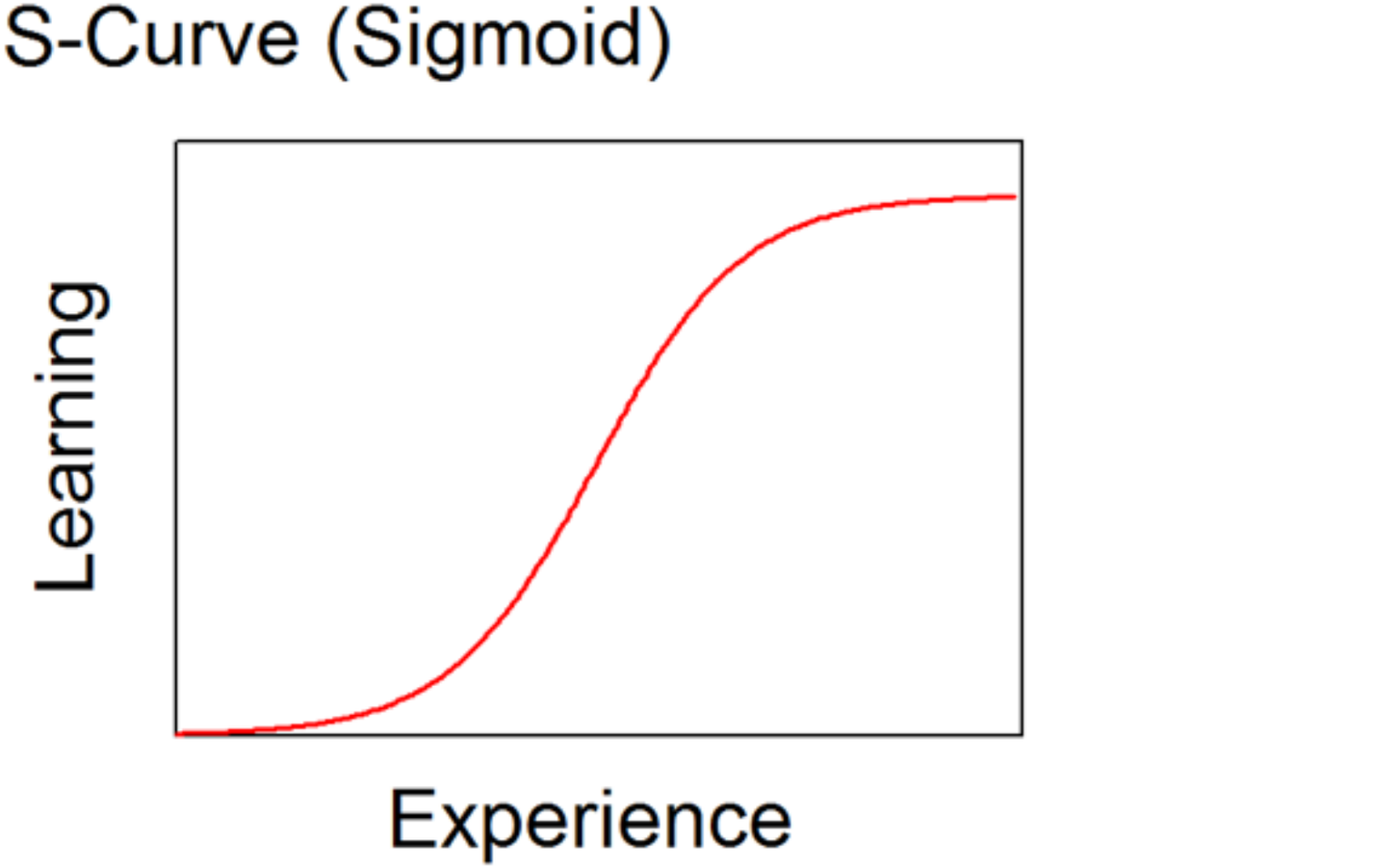
You could fork the repo, make a change, and submit the fix back to the main repo: Processing docs

GitHub

And, of course, you can share your own code and experience as well as work on a collaborative project with your friends.



With anything, there is a learning curve.



git was developed by software engineers

The basic concepts are simple, but the details can become overwhelming if something isn’t working right.

Merge conflicts are one of the harder things to get a hang of when starting.

A ***conflict*** happens when two different commits change the same line(s) of text and cannot be automatically merged together.

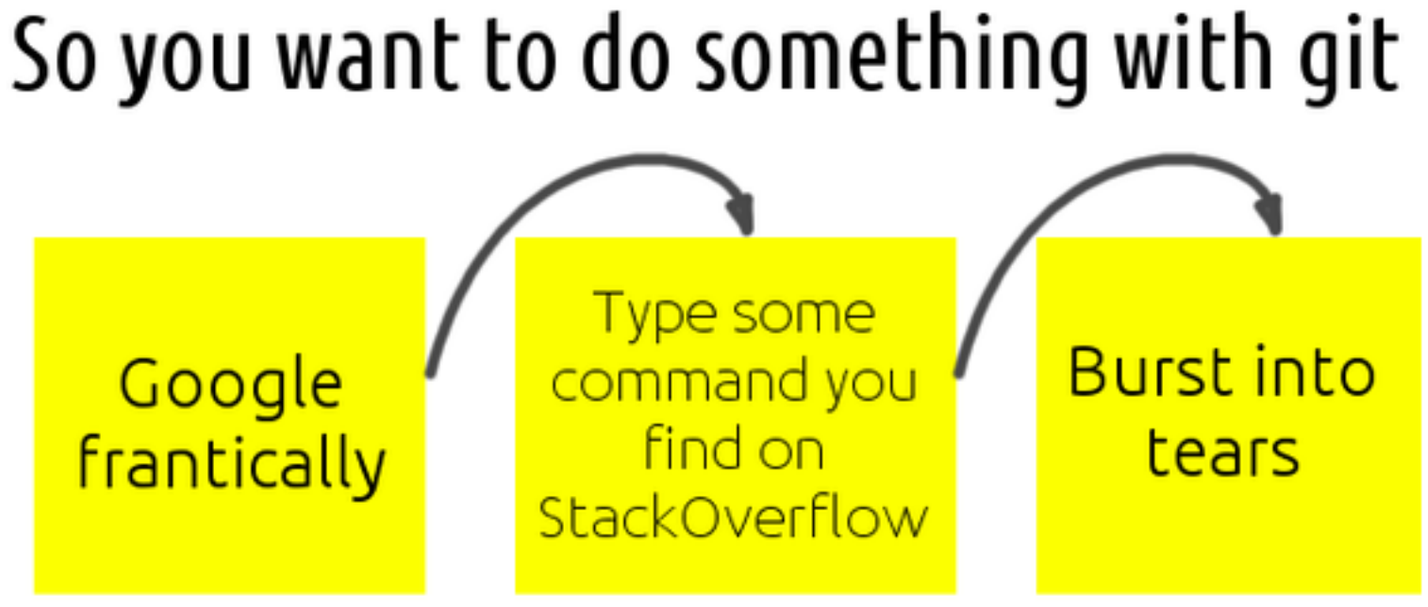
You basically have to go in and manually fix those lines. It's **not** *hard* but it can be scary the first time. GUI tools help here.

README.txt

```
1  Hello World!
2
3  Knock, knock.
4  Who's there?
5  Git.
6  Git who?
7  <<<<<<< HEAD
8  Git this joke over with.
9  =======
10 Git on with the assignment!
11 >>>>>>> master
12
```

You are not alone!

Due to it's popularity, there is *lots* of info online: tutorials, FAQS, forums, etc



A few resources:

- git - the simple guide
- Try Git - Code School Tutorial
- Intro to git & the command line
- git Documentation

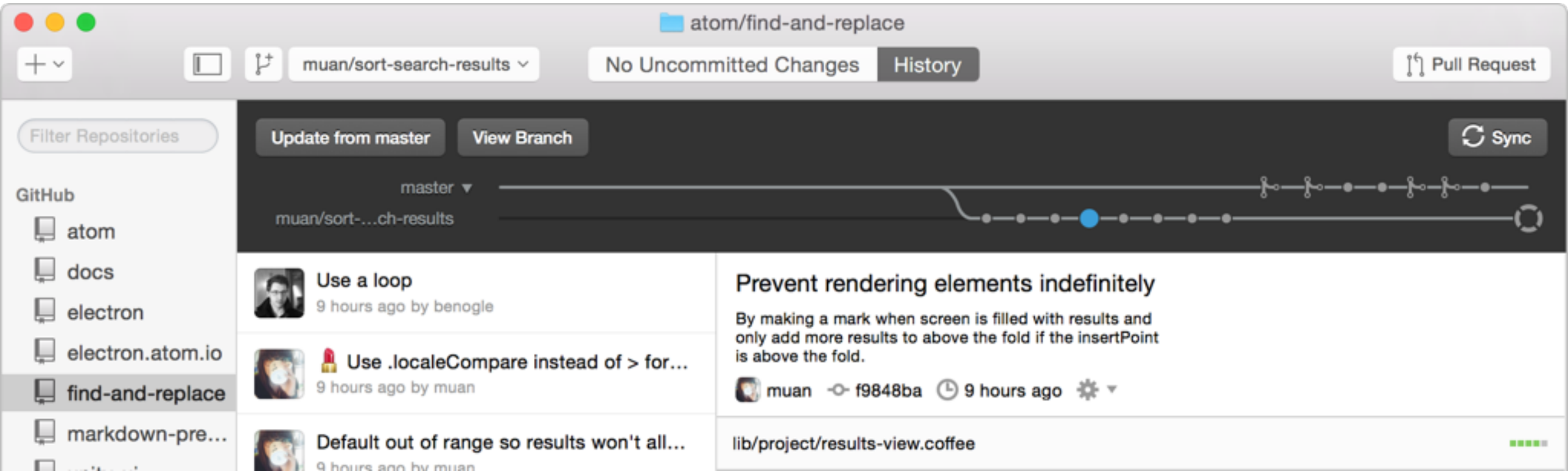
Graphical Front Ends

You don’t have to work on the command line if you don’t want to.

There are a number of graphical wrappers for git: Git SCM GUIS

GitHub Desktop is a good choice.

It also works with non-GH repositories.



The PROS outweigh the CONS

The freedom to experiment & work collaboratively will open doors. You'll be very happy when you know you have a backup every time you make a commit.

… especially in the middle of the night.


git + github
SOCIAL CODING
USING GIT
HOW IT FEELS
memegenerator.net