STARTUP FLOW

Get your team up and running as quickly as possible.

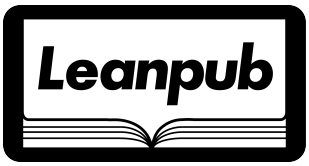
Louis Chatriot / Stanislas Marion / Charles Miglietti

Startup Flow

Get Your Team Up and Running as Quickly as Possible

©2012 Louis Chatriot, Stanislas Marion, Charles Miglietti

This version was published on 2012-05-15



This is a Leanpub book, for sale at:

http://leanpub.com/startupflow

Leanpub helps authors to self-publish in-progress ebooks. We call this idea Lean Publishing. To learn more about Lean Publishing, go to: http://leanpub.com/manifesto

To learn more about Leanpub, go to: http://leanpub.com

Contents

Preface

Who Is this Book For?

If you are a **small team beginning a technical entrepreneurial project** and you are struggling to organize your workflow, especially if some of you are working remotely, this book is for you. It was written with both business people and developers in mind as they'll need to use common tools, however there are some more *technical* chapters which are marked as such so that business people can skip them.

Why You Need this Book

You need it because **it will save you a ton of time and make your workdays much more pleasant**. Indeed, when you start working on a startup you quickly realize that you are going to need some tools to organize your workflow. Especially if you have cofounders, you will find yourself meeting too often, repeating yourself, having a hard time following what everyone is doing, and this will be very frustrating for all of you.

In order to fix these problems, **you'll need to choose your tools and design an efficient workflow** through trial and error. **That takes a lot of time**, given how many tools there are and how hard it is to discriminate between them without actually trying them. **We already went through that process, and we compiled our findings** in this book so that you can skip that part and start getting things done with great tools that will boost your productivity. Some of these tools solve issues that you might not even be aware of yet.

What Is in this Book?

This book is a collection of the **opinionated choices** we made. We focused on what we think are the most important components of a startup workflow:

- Communication
- Knowledge sharing and reuse
- Codebase management

With these 3 themes in mind, we offer a selection of tools, one tool for one problem. The tools that we promote are the ones that we use extensively everyday and that we chose after weeks of iteration. Furthermore, they all work very well together. We will describe these **synergies** at the end of the book, so that you can take full advantage of your work environment.

Communication

Communication between teammates is a hard problem. It is necessary yet so complicated to get right. A large portion of communication should be asynchronous, enabling you to avoid useless interruptions that hurt your and your teammates' productivity. You also need to use different channels dedicated to different use cases.

Knowledge Sharing and Reuse

All the knowledge that one finds interesting and usable should be findable by the other teammates, without him having to send a link by email or Skype every time. If knowledge is not shared or not reusable, it's worthless and you will lose a lot of time. This is especially true for Eric Ries' validated learning[1], on which the whole team should be constantly focused.

Codebase Management

In a technical startup, coding is a vital activity. The developers need to be able to code effectively on their own, but also to collaborate seamlessly. They need to be able to review one another's code to ensure overall quality. Teams that don't do that will hit a wall.

If you do all these 3 things right, your team will be so efficient it will be able to build the next Instagram in a week. No kidding!

What this Book is Not

This book is **not** a collection of tutorials. We will not guide you through the process of installing and using everything in details. Instead we point you to quality external resources.

It also doesn't pretend to be the *be-all, end-all* guide on team workflow for every case imaginable. However we feel it is the absolute best for small teams working on a tech project, especially if the team doesn't have an office yet.

And, finally, it is not a startup guide. We won't give you any advice as to how to get your business off the ground since we're not qualified for that! If that's what you're looking for, we recommend Eric Ries' *The Lean Startup*[2], a must-read in our opinion.

Why We Wrote this Book

As we started working on ideas and on our startup, we faced the problem we cited earlier. Our lack of organization was holding us back. So we spent two weeks researching and trying different tools.

[1] http://www.startuplessonslearned.com/2009/04/validated-learning-about-customers.html

[2] http://www.amazon.com/The-Lean-Startup-Entrepreneurs-Continuous/dp/0307887898/ref=sr_1_1?ie=UTF8&qid=1334839735&sr=8-1

After a few iterations we finally reached a very efficient workflow and this investment is now paying off big time. At first this was just an internal effort, but judging from the sheer number of questions[3] you[4] can[5] find[6] on Quora[7], we understood that many people could benefit from our research, and decided to write this book.

How to Contact Us

This is the first version of this book and any feedback that can help improve it will be genuinely appreciated! You can send us an email at book@needforair.com[8] or tweet us @NeedForAir[9]. You can also check our blog[10] out.

Acknowledgments and Inspiration

We would like to thank our reviewers who helped us improve this book:

- Charles[11] for his precise and constructive feedback
- Clement for his early support and his key remarks on the introduction
- Jeff[12], who managed not to fall asleep spotting typos during a flight
- Jonathan[13] and the Vidal[14] R&D team for their very comprehensive and detailed review of the book
- Martin[15] for his thoughtful advice on marketing and audience building
- Rand[16], for being the quickest to give us feedback after receiving our email. And it was insightful too!
- Raphael[17] for is business-oriented eye

[3] http://www.quora.com/Project-Management/Which-is-the-best-project-mangement-time-mangement-tool-and-why
[4] http://www.quora.com/What-is-the-best-way-for-a-startup-distributed-team-to-handle-document-management
[5] http://www.quora.com/Which-Project-Management-tools-are-most-useful-for-a-small-startup
[6] http://www.quora.com/What-are-the-best-productivity-tools-for-entrepreneurs
[7] http://quora.com
[8] mailto:book@needforair.com
[9] http://twitter.com/#!/NeedForAir
[10] http://needforair.com
[11] http://twitter.com/#!/Gorintic
[12] http://twitter.com/#!/jfpetitniv
[13] http://twitter.com/#!/un_john
[14] http://www.vidal.fr/
[15] http://twitter.com/#!/martinpannier
[16] http://twitter.com/#!/randhindi
[17] http://www.linkedin.com/pub/raphael-de-talhouet/15/572/a7a

- Safta[18] for her nice non-geek point of view

We would also like to thank Jason Fried[19] and David Heinemeier Hansson[20] for their book *Rework*[21] which helped us and inspired us tremendously. We recommend you read it!

[18] http://www.linkedin.com/pub/safta-de-hillerin/47/711/a57
[19] http://twitter.com/#!/jasonfried
[20] http://twitter.com/#!/dhh
[21] http://www.amazon.com/Rework-Jason-Fried/dp/0307463745/ref=sr_1_1?ie=UTF8&qid=1334839838&sr=8-1

Communication in a Small Team

Internal communication is hard to do efficiently, and bad communication will dramatically slow you down. Let us describe a few concepts that are the foundations on which we chose our tools.

Synchronous and Asynchronous Communications

Communicating synchronously means that all the interlocutors are communicating at the same time. This is why it is called synchronous. This is usually the case for all voice-based communications, such as face-to-face conversations, phone calls and conference calls. In a synchronous conversation, people expect an instantaneous response from the other participants.

On the other hand, asynchronous communication entails letting people take part in the conversation when they want. That could be a few minutes later, a few hours later, or even a few days or weeks. For that to work, asynchronous communications need to rely on a written exchange. It is not possible to have an oral conversation where inputs arrive every few hours. They just don't work with delay. Examples of asynchronous communication media are letters, text messages, emails, chat or carrier pigeons.

Most people and teams use both forms of communications naturally, but often applied to the wrong usecases. It is not rare for instance to see people send an email and expecting a quick answer, or conversely to see people engage in oral conversations that really should have been asynchronous because the matter is not urgent. In practice, long conversations tend to take the shape of meetings which are notorious productivity killers, while if done in an asynchronously they can yield better results and be less disturbing at the same time.

Most Communication Should Be Asynchronous

The massive use of text messages illustrates the usefulness and the advantages asynchronous communication offers over an oral conversation. Itâ€™s quick and efficient, one can respond hours later and keep track of history. In most cases your communication with your teammates should be asynchronous for the following reasons.

Asynchronous communications don't disturb people. By sending a message to someone, you let her read it and respond when she sees fit. You don't bother the person you are communicating with, nor the other people around you that might overhear your conversation if you're talking in an open space for instance. Distracting your teammates strongly hurts your productivity as a team and is conducive to tensions. Furthermore, synchronous conversations have a high tendency to drift completely off-topic, whereas their asynchronous counterparts are more stable.

Asynchronous communications being written, they are stored somewhere (chatlogs, emails, etc.) and you can go back to it later, to refresh your memory or to bring someone new to the debate. This

is important because humans are inherently bad at recalling conversations in an impartial way.

Most topics of communications are not urgent, and most shouldn't require the focus of two or more people at the same time.

When to Be Synchronous

There are a few cases though when asynchronous is just wasting time and won't cut it. Here they are:

- Urgent matters
- Crucial decisions that need lengthy explanations and argumentations. These should be very rare in practice.
- When interlocutors are effectively chatting to each other in synchronous mode and the discussion is going nowhere. In that case it usually feels obvious a vocal conversation will be more efficient and the best thing is to pick up the phone or start a Skype call to settle the matter.

Segmentation by Use Cases

Being asynchronous is not enough. If you don't compartmentalize your conversations by use case, you'll end up using email for everything, and we all know how this ends. Instead you should have different channels for different topics. We have a chat room for everyday conversations (and subchannels inside the chat), Github for code-related conversations, Trello for project management conversations, and email for reports and formal announcements. That way it becomes easy to follow on everything that's going on, and you don't face a mountain of emails everyday. We will detail each of these channels in this book.

How to Use this Book

The workflow made possible by using all these tools is great but we are aware that learning 10 new tools from scratch can look daunting. It is probably better to take a step-by-step approach and appreciate the benefits incrementally. As such, you should probably read one chapter that resonates with your current workflow pains and try out the tool we suggest. We however strongly insist on the fact that the whole package is incredibly powerful, and it would be sad if you didn't try it out!

Some of the tools we recommend are super intuitive and will enhance your productivity by a big margin very quickly. Others have more friction and a steeper learning curve. In order to help you start, we list some of the quick wins that you can try out first, and then we explain what will demand more time in order to achieve their full potential.

Technical and Non-Technical

Some of these tools are clearly meant for developers. They are tagged as technical. For reference, here is the list: Operating System, Git, Github, Vim and Synergies.

Quick Wins

- Trello and HipChat are both extremely simple to set up and to use. If you are looking for quick wins in the project management and communication spaces, you should start by reading the articles concerning these two tools and try them out as soon as possible.
- Basic Git commands are simple and will give you 80% of Git's power. It shouldn't take much time for you to learn them and you will immediately feel the benefits.
- Github enables you to easily use a central Git repository. It is easier than having to set it up yourself and will save you a lot of time. There is also a Github application for Mac OS that will set up Git for you and give you a graphical interface that is genuinely well designed. You should be able to bootstrap your learning of Git and Github with great efficiency thanks to this app.

Long-Term Investments

- Thoroughly understanding Git is paramount. Even though becoming an advanced Git user will take some time and dedication, it is worth it when you are faced with a problem or want to have more fine-grained control over your codebase.

- Github will bring a huge boost to your productivity and creativity once you start using pull requests[22]. As soon as you're used to working with a centralized Git repository, you'll definitely want to look into pull requests and incorporate them in your workflow.
- Vim is a super powerful editor. But it's true the learning curve is steep. We recommend learning Vim once your workflow is in place and you already improved your productivity so much that you can afford to spend the time learning Vim.
- Building your wiki and your knowledge database with Evernote is essential to capitalize on your everyday work. You won't necessarily see the benefits straightaway but you will weeks or months later.

Goldmines of Help and Information

For all the questions this book doesn't answer, Stack Overflow[23] and Quora[24] are the websites you should go to to find answers.

Recap

This table offers a synthetic view of the tools we recommend.

Chapter	Tool	Learning Curve	Technical
Project Management	Trello	Easy	No
Chatroom	HipChat	Easy	No
Email	Fiesta.cc	Easy	No
Version Control	Git	Medium	Yes
Collaborative Coding	Github	Medium	Yes
Knowledge Sharing	Github Wiki	Easy	No
Editor	Vim	Hard	Yes
File Sharing	Dropbox	Easy	No
Content Sharing	Evernote	Easy	No
Expenses	Tricount	Easy	No

[22] http://help.github.com/send-pull-requests/
[23] http://stackoverflow.com/
[24] http://quora.com/

A Day in the Life

Before diving into the actual explanations about our choices of tools and in order to put the rest of the book in perspective, we would like to give you an overview of how we work as a team, by describing one of our typical workdays. Of course, you should not yet know all the tools we mention here, but this should still give you a sense of how they can be used together. Here it is:

- **Apr-21 01:53 AM** | Charles just finishes his work on the *SecretPhotoSharingApp* project. He pushes it on a new branch named `killer-feature` on Github and opens a pull request to signal it needs a peer review.
- **Apr-21 07:45 AM** | Louis wakes up and sees the Github notifications concerning Charles' work in the hooks-dedicated HipChat room. He clicks on the link in HipChat to open the pull request and starts reviewing Charles' code. There is some good stuff and some things that need to be changed, so he commits some fixes and pushes them. He explains his reasoning in the pull request. No need to write an email, Charles and Stan will be notified through Hipchat.
- **Apr-21 08:20 AM** | Louis starts writing a blog post. He wants to talk about politics but he isn't sure that the subject will interest their audience. He begins writing a first draft.
- **Apr-21 08:30 AM** | Charles wakes up -he doesn't need much sleep- and starts reading some interesting articles that popped up on Hacker News and Twitter. He decides to bookmark and share some of them with Evernote and starts a conversation on HipChat. As Louis is working and Stan is sleeping, this conversation will only be continued a few hours later, which is completely fine.
- **Apr-21 08:45 AM** | Louis just finished his blog draft, so he pushes it to a new branch in the blog repository on Github, and opens the corresponding pull request to collect feedback from Charles and Stan. He then begins to analyze the results of a test -a tentative new "purchase" button- he had launched a few days ago on *SecretPhotoSharingApp*.
- **Apr-21 10:30 AM** | Stan wakes up late as usual, checks on the notifications, and thinks that Louis' amendments to Charles' code are fine. He says so in the pull request.
- **Apr-21 10:45 AM** | Stan feels like he'll go on a coding spree this morning. He fires up MacVim to resume working on `killer-Feature-25`, another killer feature the team doesn't want to ship the product without. He's been working on it for the past few days in parallel to Charles' and Louis' progress.
- **Apr-21 11:00 AM** | During his break from coding, Louis reads some of the articles Charles shared. One of those contained a useful but hard to remember Git command, so he writes it down in the wiki's Git cheatsheet for future reference.

- **Apr-21 11:00 AM** | Charles catches up to Louis' fixes and Stan's comments to his code. Since `killer-feature` is done and has been reviewed, he merges the pull request: our *SecretPhotoSharingApp* has now a new `killer-feature`! He also removes the "build Killer Feature" item from the Trello board to reflect the work done. He begins working on `another-feature`.
- **Apr-21 11:30 AM** | Stan has a quick question for Louis about a piece of code the latter wrote. He asks it on the private chat channel so as not to disturb Charles.

…

And so on and so forth! After a review from Stan and Charles, Louis' article will be posted on the blog.

You may have noticed how we didn't need any meeting or phone call. We didn't even need to send an email, as there was no important issue that day. We switch tasks only when we want to, so we can really focus on our work without being disturbed.

Project Management - Do It Like a Pro

Todo Lists

When working on a project, you often think about a new idea or a bug to fix while doing something else and you want to keep track of all these. That's why todo lists were invented. Todo lists are so simple and modular that you can organize and manage all your work with them. For managing our short, middle and long term workload we chose to use Trello[25].

Why Do We Use Trello ?

The common problem with todo list software is they actually end up getting in the way of real work. The classic tragic ending is when people are happy because they just prioritized and assigned 50 bullet points without getting any work done. Two weeks later the todo list is abandoned because it's too much work to maintain.

But Trello is different. We tested Asana[26], RememberTheMilk[27], and simple google docs. None of them survived more than a week. Trello has survived for more than 2 months now and is still going strong. The reason is that at heart Trello is a simplified and collaborative version of Excel: you get columns and cells, but with a fantastic user experience. You'll see that people are naturally used to that kind of visual data-structure and will use it effectively. You can catch a global overview of the progress of your project in one glance. So throw away those 23657 post-its that have been sitting around on your desk for 2 weeks and start using Trello now!

How Does Trello Work ?

On Trello, you create a board (just like a physical one), accessible by all teamates from their computers and phones. Each board is composed of a certain number of lists. On each list every teammate can add items (called 'cards' that are just like post-its), move them within the list or across lists with a simple drag-and-drop, assign them to one or several persons, and archive them when they're taken care of. Notifications are sent for each modification to keep you updated.
What really makes Trello user friendly is its two layers of information. By looking at a board you have all the top level information you need to keep track of the work in progress. If you need more details on a task you can zoom in to the card view where you may add comments, have a discussion with your teammates, browse the card history and so on.

[25] https://trello.com/
[26] http://asana.com
[27] http://rememberthemilk.com

Further Information

Trello's guide[28] is great for getting started.

Real Life Example

We use Trello Boards for this book, our product and our blog[29]. Let's detail the process we used for this book. At first we jotted down our ideas for the tools we wanted to cover. We created a `Nothing done` list and a card for each tool. Then we created the following lists: `Ready for Review`, and `Reviewed`. Any time one of us started working on a new chapter, he would assign himself to the card so that it was easy to see what was being worked on at any given time, and what was left to do. Once the first draft of the article was done, the person who wrote it would move the card to the `Ready for Review` column. The other would now review the chapter and assign themselves to the card once the review was done. Once we agreed on a final version for a chapter, the card was moved to the `Reviewed` list. Any time someone had a question or advice on an item that was still in the `Nothing done` list, he could ask a question or write a piece of advice in the card. Further discussions took place on Github pull requests. This very simple process worked wonders.

For those interested in a usage example in a larger and more mature startup, Uservoice[30] published a great blogpost[31] that details their product management process.

[28] https://trello.com/guide
[29] http://needforair.com
[30] http://uservoice.com
[31] http://www.uservoice.com/blog/founders/trello-google-docs-product-management/

Chat Room - Stay Connected

Your Chat Room = Your Living Space

You can't live without a chat room when working as a team. You need to be able to communicate seamlessly with your coworkers. This is true even when the whole team works in the same room: if you say something out loud, you might break your coworker's flow and focus. She has to answer right away instead of when it suits her best, which can be very annoying for her. This is the reason why we started using Skype chat when we were in the middle of our freelance mission for the French Employers Association. Even though we were two meters apart, we almost never disturbed one another. We really felt the productivity boost when we stopped asking oral questions all the time!

Another pitfall of oral conversations is that they won't be logged anywhere. What if it ended up being important or interesting for another person? It's gone and that's too bad.

So the main reasons for using a chatroom are:

- Conversations are asynchronous: people answer when they see fit and don't have to shift their focus away instantaneously
- Conversations are logged, meaning you can go back to it later, invite someone to read it, build upon it later, etc.
- You need less meetings
- Remote work becomes way easier

Why not Skype? Because of HipChat

Skype sucks in the way it keeps history. And it has a really bad user experience for group chats and channels.

HipChat is great for all of these. You can create as many channels as you want, have one-on-one chats as well as group chats. The history of conversations is easy to follow and searchable. On top of that it offers you a slick desktop app, and we found this was a much better experience than having to chat in your browser. You can chat on the go as well thanks to their iphone app.

Did I mention it is a hosted solution? You have almost nothing to set up. No need to try and set up IRC on your server and then struggle to keep it secure, and log it properly. Your time is better spent building your product!

Set up HipChat

To set up HipChat you need to register[32] first. Then you will be able to invite the remaining collaborators to join. Download the desktop application for a better user experience.

Price

Yeah you're a startup and you're on a budget we know that! But at $2/month/account, it is a steal and the better communication and saved time are more than worth it.

Real Life Example

We created rooms for the following topics:

- **Work**: where we discuss anything related to our work that needs the 3 of us to participate. These are supposed to be short conversations, we're not going to discuss our 10-year roadmap like this.
- **Lounge**: where we discuss anything not directly related to our work, such as recent articles, tech news, traffic on our blog, when is the next squash session, cat pictures and what we've had for dinner.
- **1-on-1**: we all have private rooms with the two others, so that we can discuss all kinds of issues without bothering the third one. It's very useful for asking specific questions or work together on something.
- **Product Hooks**: where we get all the push notifications about our product from Github about commits, pull requests, issues, etc.
- **Book Hooks**: same but for this book's repository
- **Blog Hooks**: same but for for our blog[33]'s repository

For those interested in a usage example in a larger and more mature startup, Uservoice[34] published another great blogpost[35] that details how they use HipChat with larger teams and their integration of client satisfaction and engineering-related hooks.

[32] https://www.hipchat.com/sign_up
[33] http://needforair.com
[34] http://uservoice.com
[35] http://www.uservoice.com/blog/founders/how-we-use-hipchat/

Email - Tame the Monster

Email is the single most used tool of all time for management, internal communications, project tracking, etc. While there are some correct use cases for email, we believe most are plain wrong. And if you do email wrong, you'll end up losing a lot of time, money and patience.

What You Should Use Email For

We all receive tens or even hundreds of emails daily, and we don't read half of them. That's why it is important that work-related emails don't clutter your mailbox and impact your productivity negatively. We only use email if the content matches the following conditions:

- It's an important report, announcement or other formal communication
- It is not bound to be centralized in a wiki in its current form and context
- It is not subject to debate or discussion. It's mostly a one-way communication.

Following these rules will prevent your inbox from getting cluttered by *"let's go eat lunch now!"*, *"check this link!"* or *"review my code"* emails, while still keeping important information such as meeting reports, the announcement of the adoption of a new tool or welcoming a new recruit. It also prevents the mess that quickly become email discussions and debates.

The most important rule when writing an email is choosing and formatting your email subject. It's the only thing the recipient will see with your name! So your email subject has to be precise, neutral, searchable and should provide a good overview of the content. Use *[AMERICAN IDOL]* formatting in your subject to create a visual categorization of your emails and setup automatic filtering.

Email Clients

While your webmails are fine (especially if you're on Gmail[36]), we recommend the use of an email client. It allows you to download and archive your messages, so that you can read them without an Internet connection. In addition, they enable you to aggregate several email accounts in the same place (in our case, we have personal Gmail addresses and needforair.com[37] addresses).

- On Mac OS X, we recommend using the simple Mail app that comes with Mac Os X, or if you want to get fancy, Sparrow[38] is lightweight, beautifully designed and focused on the messages. It eliminates all the fluff of a client such as Outlook.

[36] http://www.gmail.com
[37] http://needforair.com
[38] http://sparrowmailapp.com/

- On Linux, we recommend using Thunderbird[39]. It is simple to setup and use, the filter system is intuitive, and, cherry on top, you can have Gmail-style conversations using the Thunderbird Conversations addon[40].

Internal Mailing Lists

As you will find yourself sending email to the same group(s) of people, adding the same recipients manually over and over will become tedious. That's what internal mailing lists are for: there is only one email address to enter, and you can add or remove members of the mailing list easily. Besides, you can use this address to filter all emails from the mailing list and see all work related emails in the same folder. You can also create a different mailing list for each topic, so that your emails are automatically sorted.

Fiesta.cc Rocks

We use fiesta.cc[41] to manage our mailing lists. We haven't tried another service, but we don't see how something simpler could be built: you don't even need to go to their website to use it! You read that right: simply send an email to the other members of your mailing list, put *theavengersteam@fiesta.cc* in copy of your email and click *send*. After that your mailing list is set up and you're good to go: any member of your list can send an email to the others by sending it to *theavengersteam@fiesta.cc*. Also, you don't need to worry about *theavengersteam* being already taken, as fiesta remembers that this is the one you created. Some other lists with the same name may exist, but won't impact yours.

Fiesta.cc offers a really neat management interface for your mailing list, through emails. We encourage you to find out more about it here[42].

Happy emailing!

Real Life Example

Recently we sent emails for the following reasons:

- To announce the migration of our blog from Posterous to Octopress and give installation instructions
- To forward some of the feedback we've been getting individually about this book

[39] http://www.mozilla.org/en-US/thunderbird/
[40] https://addons.mozilla.org/en-US/thunderbird/addon/gmail-conversation-view/
[41] https://fiesta.cc/
[42] https://fiesta.cc/learn

- To inform one of us that was on holiday of a big decision we made while he was sipping mojitos under the sun.
- To say we're going to lunch. Nah just kidding!

We very rarely use email.

Operating System - Build on Solid Foundations

***If you are not a developer, keep the OS you are used to**, all three of them are fine.*

The first choice when building your work environment is the choice of your operating system. It's a 3-options choice: Mac OS X, Linux or Windows.

Mac OS X and Linux: A Good Fit for a Developer's Needs

We recommend that developers use a UNIX-based system, be it Mac OS X or Linux. They really feel like they were designed for making software development a great experience. The three main reasons are:

- The Terminal (see below)
- Package managers (see below)
- The fact that most servers run UNIX-based system, so it is best to be used to them and not have to switch often between two very different OS

Keep in mind though that **there are specific cases where you will want to use Windows** such as:

- You need Microsoft's .NET platform, Windows will obviously be better suited
- You make desktop applications targeted to businesses which mostly run Windows on all their machines.
- You want to play the latest videogames!

The Terminal

This is the main reason why we prefer UNIX-based systems. The terminal enables you to give commands directly to your computer, and is used for a variety of purposes: installing/uninstalling software using the package manager (see below), managing your files' versions (see chapter dedicated to Git), compiling your programs and so on. It also enables you to fully manage a remote server. If you don't know how to use it or if you are afraid to use it, it's time to grow a pair and make the big leap. Take Zed Shaw's crash course[43] and master it! You will dramatically improve your productivity on your computer.

You can find terminal emulators for Windows, the most well-known being Cygwin[44], but they are a pain to use. One of us tried and gave up quickly.

Package Managers

Packages are utilities and tools that you need when you develop software. To manage them, you need a good package manager[45] or things get real messy real quick. Installing and uninstalling a package is made as simple as installing an app on your smartphone. Apt-get[46] for Linux and Homebrew[47] for Mac Os X are the package managers we use. Windows also has package managers[48], but they're not widely used so you will often need to install packages manually, which takes more work. They're also much less practical.

I Need to Choose! Mac OS or Linux?

Two of us are running on Mac OS X and one on Linux, and we're all satisfied. On the one hand, Mac OS X is easier to use and comes with many tools preinstalled, and most things just work on it. The graphical interface is gorgeous. On the other hand, Linux gives you more control on your computer, and is installed on most servers, including the ones you'll need, so knowing Linux will prove valuable in the long run.

If you want to use Max OS X, you'll need to buy a Mac (go figure!). If you want to give Linux a try and install it on your PC, we recommend the easy-to-learn Ubuntu[49]. You can download it and install it using a USB stick.

[43] http://cli.learncodethehardway.org/book/
[44] http://www.cygwin.com/
[45] http://en.wikipedia.org/wiki/Package_management_system
[46] http://en.wikipedia.org/wiki/Advanced_Packaging_Tool
[47] http://mxcl.github.com/homebrew/
[48] http://en.wikipedia.org/wiki/List_of_software_package_management_systems#Microsoft_Windows
[49] http://www.ubuntu.com/download

Version Control - Safely Iterate on Your Code

This is probably the **single most important tool** you will need apart from your brain, your hands and your computer. You might not realize it now if you never had to use a version control tool before, but you'll see.

SVN users, don't skip this chapter! It explains **why Git is genuinely better than SVN**.

Version Control, What Is It ?

When you learn how to code, you start naming your files like `my-cool-program`. Then you want to make it better, but you are scared to overwrite the previous version because it might not work. So you wisely name your new version `my-cool-program-v2`. You show it to a friend of yours who thinks what you built is really cool and he is going to try and make it even better! So you email him the file and the next day, lo and behold, you get a `my-cool-program-v3-reviewed-by-friend` in your inbox. What if you want to accept only a few of his modifications? What if after doing this you want to rollback to your v1? It gets really messy real fast and then it gets worse, and then worse again.

So then you think there must be a better way to do this mustn't it? As any good programmer, you want to automate stuff so you can focus on your code instead of managing versions.

The good news is that there are tools to deal with this problem. They are called Version Control Systems (VCS). Using a VCS allows you to experiment with your code while always keeping a stable version easily. They allow you to show your teammates and the world what you worked only when it is ready, without messing up the program for them in the process.

The Ultimate VCS: Git

Git has been invented by none other than Linus Torvalds[50] a few years ago, and is the most elegant and convenient solution out there. It is straightforward, easy to use and incredibly powerful. The good thing is that the learning curve is very smooth. You can start using it for simple tasks at the beginning and learn more complex stuff as you go.

[50] http://en.wikipedia.org/wiki/Linus_Torvalds

Installing Git

You will find simple installation instructions here[51]. You can either install from source, with a package manager or with an executable, depending on your OS.

How Does Git Work?

Basically what Git does is track modifications of your files and allow you to take a snapshot of the current state of your files at any given time. When you have a number of snapshots, you can navigate between them, and follow the history of your code.

The inner workings of Git are well explained in the Pro Git book[52]. You should read the first three chapters to understand how a modern VCS works. You will learn what a repository, a commit, a branch, a merge, and a checkout are. After this, you will be ready to start using Git, and soon you'll feel really comfortable with it. **It is crucial that you read these three chapters to understand what version control is and how it works**. This won't be a waste of time: as you begin to use Git's more advanced commands, you will find more than helpful to have a real understanding of what you are doing.

Git versus SVN

This paragraph is geared towards SVN users, not version control beginners

If you have used SVN before and you're wondering why you should switch to a new system, then here's why:

- Branching[53] in Git is simple, natural, fast and makes for great unobstrusive collaborative workflows
- Merging is a breeze: ever had problems with merging in SVN? Having to choose between a reverse merge, a branch merge, a something else merge in TortoiseSVN? In Git, merge just works, another reason to go on a branching spree!
- You can work and commit **without having to be connected to the server**. I'm not kidding, you're not dreaming. Yes, you really can keep track of different versions on a plane!
- The CLI is fairly straightforward and lightweight
- Github[54] was designed to work with Git and makes your team even more productive

[51] http://progit.org/book/ch1-4.html
[52] http://progit.org/book/
[53] http://en.wikipedia.org/wiki/Branching_%28software%29
[54] http://github.com

Feature Branch Workflow

Scott Chacon, author of Pro Git book[55], and other fine folks working at Github[56] recommend a workflow[57] based on feature branches. Basically one starts a new branch from master every time one wants to develop a new feature. This allows to keep an always deployable master branch, and always have a clear view of what features are being developed. It also mitigates the risk of two branches overlapping in scope and ending up conflicting.

Real Life Example

Charles wanted to build the *handling PATCH requests* feature for our API. He created a branch named `handle-patch-request` and started coding. After some time, there was an urgent bug *some special characters not displaying properly* to fix, so he simply commited what he was working on, switched to the new feature branch `hot-fix-special-characters`, fixed the bug and pushed to `master`. He then resumed his work on the *handle PATCH requests* feature. Meanwhile, Stan reviewed his hot fix and merged it into `master` to make it go live right away. It really was that easy.

[55] http://progit.org/book/
[56] http://github.com
[57] http://scottchacon.com/2011/08/31/github-flow.html

Collaboration on Code - Unleash the Power of Your Team

Making software as a team is not only about writing code, it's about writing good code that your present and future teammates will understand. In order to write good code, you need efficient and rock-solid yet unobstrusive processes. You need a way to cleanly and naturally communicate around your code.

And you need anyone to be able to understand what's going on with your codebase and start contributing to features they didn't start without having to ask too many questions.

You also need a place to centralize all your work because you don't want your different projects to be scattered around.

Using a cloud service to host your repositories is a good idea instead of hosting them on your own server, because it's faster to set up, manage and is probably going to be more secure.

You Already Met Git, Now Meet Github

Github[58] is the sweetest web-based interface you'll ever encounter for hosting and working with your Git repositories.

At its core Github is a platform where you can centralize and follow all your team's coding activity painlessly. The activity feed and notifications enable all your team to follow independently what's going on without having to ask any questions and disturb anyone. It really changed the way we approached managing a software project and is the crux of our entire workflow. Without Github we would definitely not be able to monitor our progress and collaborate so seamlessly and efficiently.

What Makes Github So Awesome?

The Pull Request System

The single most important thing that Github enables with its interface is having conversations on every new (or old) feature of your code. And those conversations follow the same architecture as

[58] http://github.com

your repository branches so you don't need to organize them yourself, search for them, decide where they should live. These conversations will be partitioned at the feature level if you follow this simple workflow[59]. Github calls these conversations issues or pull requests. A pull request is simply an issue with code attached that needs to be merged, whereas issues can be more general discussions. The pull-request based workflow morphs naturally around feature branches, which we talked about in in the Git chapter.

We can't stress enough how well thought-out the conversations interface is. First it's asynchronous. People will take a look when the time is right. Second, it incorporates everything there is to know about an impending change of code: all commits and diff comments appear inline, making it effortless to follow and get up to speed. Third, it gives you a unique channel of communication for all your code conversations. That is hugely efficient. The code and the conversations around it are in the same place. No need to check emails or chatlogs. It's like a forum where each feature is a thread and the thread follows the code it is about, and that allows anyone at anytime to understand why a certain feature was implemented the way it is.

At first these can sound like a lot of metawork, but seriously pull requests will save you so many boring meetings and email trouble that they're completely worth it. More than that, they're actually really nice to work with.

The Community

The interface is so good that Github has been enjoying a tremendous success and a large number of large high-profile open-source projects are now hosted there, such as Linux, Rails, NodeJS and Twitter Bootstrap. But it doesn't stop at high-profile. An overwhelming number of smaller but quality open-source are hosted on Github, many more than on other code hosts such as Gitorious[60] and Bitbucket[61]. In that respect it is very useful to know their interface well, so that you can discover and browse projects easily. Having a Github account makes it easy for you to connect to the developers of an open-source library you use. They are usually very quick to answer and helpful.

Bear in mind as well that since Github is now widely adopted by software engineers around the world, it is likely that the people you'll want to hire already know how to use Github. In fact they probably already have an account, and looking at it is a much better check than giving them the FizzBuzz[62] test!

Its Interface

Github is well designed, and its usability is close to perfect for such a complicated interface. Without going over the whole list of what we like, we especially enjoy:

[59] http://scottchacon.com/2011/08/31/github-flow.html
[60] http://gitorious.org/
[61] https://bitbucket.org/
[62] http://www.codinghorror.com/blog/2007/02/why-cant-programmers-program.html

- The pull request interface we mentioned before, and which makes code reviews and iterating on new features a breeze
- The network graph[63] which shows all the branches of your repository

It's hard to pinpoint features in particular, because what we really like is the entire user experience; the only way to understand this is to actually use it!

Cherry on Top for Mac Users: the Github Mac Application

Github developed a Mac app to provide a clean and functional graphical interface to Git and Github. It is genuinely well designed in the sense that it lets you start using Git and Github without knowing about Git commands and takes care of some annoying set up technicalities such as SSH keys. It also makes some of the more irritating parts of dealing with a remote Git repository a breeze. Starting out with the app is a great way to get acquainted to the Git and Github, however we strongly urge you to learn more about Git and especially learn its command-line interface.

Getting Started

Github is straightforward to start working with. Their Getting Started guide[64] is very clear. You can use basic features only if you wish, this will save you the trouble of hosting and maintaining a Git repo on your server, but where Github really shines is in the organization of your coding workflow and we highly encourage you to adopt this one[65].

Real Life Example

Consider this very chapter. It was collaborated on via Github. The first draft was created by Stan, who created a feature branch named `github-chapter`. He then pushed the draft to Github and opened a pull request, asking for a review. Louis saw the notification 30 minutes later and reviewed the article, fixing some typos and rephrasing some parts. He also wanted to make a big change, but wanted to talk about it first. So he simply asked his question and detailed his arguments in the pull request. Two hours later, Charles saw the pull request and read through it. He thought Louis' idea was good and implemented it. Later, Stan went back to the pull request and could see all the commits and comments. In 5 minutes he was completely up-to-date on the progress. Agreeing with Charles' and Louis' changes, he finally merged the pull request into the master branch, making the chapter part of the book.

[63] https://github.com/blog/1093-introducing-the-new-github-graphs
[64] http://help.github.com/mac-set-up-git/
[65] http://scottchacon.com/2011/08/31/github-flow.html

Knowledge Sharing - Centralize What You Know

When you're working on a project you want to build on your work. The best way to do it is to write down all the knowledge you gain regarding the project. Doing this you will structure it, sharing it will be smooth and you will able to come back to it later. A good knowledge sharing and reuse tool is paramount in order not to lose any valuable piece of info.

Why do You Need a Wiki?

You've probably heard of Wikipedia[66]? A wiki is simply a smaller version of Wikipedia, where all teammates can create and edit pages.

Wikis are the best tools for sharing domain knowledge, i.e. everything that is written internally directly in relation to your project rather than resources that you find on the web. It is not the same kind of knowledge that you'd bookmark.

The objective is to diminish the number of questions that one teammate has to ask another about something that concerns the project, be it vision, technical aspects or user feedback.

Also, a new recruit will get up and running quicker with a good wiki.

We use them to store a lot of data about our projects:

- Validated learning[67] we acquired on our startup
- Conversations with other people about our projects and the insights gained
- Our core values
- Code documentation
- Best practices
- Technical shenanigans that are particular to the tools and programming languages we use but that we might forget (arcane Git commands for example)

[66] http://wikipedia.org

[67] http://www.startuplessonslearned.com/2009/04/validated-learning-about-customers.html

The Return of Github

We didn't tell you everything about Github. In fact, not only do they make your codebase management a breeze, but they also provide you with other tools, and among them, a super simple wiki.

Github's wiki is great because it's just another Git repository, and everything is written in Markdown, which is a lightweight markup language that's easy to learn and use. And since it is a simple Git repository, exporting your data is dead simple and you don't have to be connected to the Internet to update it.

Furthermore, hosting is provided by Github, so you don't have to worry about that.

It is more lightweight than for instance Mediawiki[68] or Trac[69], and the fact that it is integrated with all the tools you're already using (git, Github) is great: no need to learn anything more.

[68] http://www.mediawiki.org/wiki/MediaWiki

[69] http://trac.edgewall.org/

Code Editor - Raw Power at Your Fingertips

Because coding will be your main activity, you will need a powerful editor to improve your productivity. First of all, it is important to realize that the speed at which you type will be the biggest factor of your productivity as a developer. Steve Yegge's rant[70] on this topic is both amusing and insightful. If you don't touch-type[71] already, the time you invest in learning it will be one of the best investments you ever make.

If you don't want to spend too much time learning a new editor but still want a great one,Sublime Text 2[72] is for you and you should stop reading this article here. However if a steep learning curve doesn't frighten you, read on.

The editor we recommend, Vim[73], is the one we found amazing. It makes us save a ton of time and increases our productivity dramatically. Two of us were used to Sublime Text 2[74] and Textmate[75] which are great non modal editors. However, when Charles showed us how efficient he was with Vim, we made the change and couldn't be happier with that decision. Vim is more lightweight, more customizable and faster. At first you will find it very counter-intuitive but don't panic, that's normal. Once you get it you'll understand how well designed it actually is. Seeing someone go to town at full speed on Vim is a beautiful thing.

What is Vim?

Vim is a modal editor, meaning keys won't have the same result depending on which mode you are in. There are different modes of execution:

- Insert mode (where you type text)
- Select mode (where you select text)
- Command mode (where you manipulate text)

[70] http://steve-yegge.blogspot.fr/2008/09/programmings-dirtiest-little-secret.html
[71] http://en.wikipedia.org/wiki/Touch_typing
[72] http://www.sublimetext.com/
[73] http://www.vim.org/
[74] http://www.sublimetext.com/
[75] http://macromates.com/

Navigation and classic manipulation of text like copy/paste, delete, find/replace, etc. are done in the command mode with the keyboard.

As a result, navigation and manipulation of text and code are a lot faster. You don't need to press backspace so much to delete words or lines. You don't have to use your mouse to select text and move it around. Every action you are used to has a mapping key so you can save time by never touching your mouse anymore. There is even a key mapping to automatically indent your whole source file! You always stay in the same window so that you always focus on your code. For more explanations on Vim, the Web is full of resources like this one[76].

Vim comes packaged in most Linux distributions and is the editor of choice to remotely edit files on a server via the Terminal. So knowing how it works will really help you when you need to maintain your server(s).

For your everyday work you'll definitely want a desktop application you can run separately from a Terminal instance. You should use MacVim or GVim.

How to Get the Hang of Vim?

At first this sounds really weird, but the fact is that after 2 weeks of mild confusion you will get more productive than on any other editor, guaranteed.

The best way to understand Vim is to see one of your peer use it live in front of you. This is the only way we've ever managed to convince ourselves and other people to try it out seriously. So our advice is to find such a person.

Furthermore, you should start with a clean and basic Vim configuration. Don't try to install plugins at first. Just learn a few basic commands every day, like Yehuda Katz[77] suggests.

You can explore the power of Vim with this wiki[78], with online[79] tutorials[80] or even while playing a game[81].

Tuning Vim

What makes Vim such a powerful editor is that it's fully customizable. With Vim you'll always find a plugin for your needs. Remember this ad *"There is an app for that"*[82]? The same goes for

[76] http://blog.interlinked.org/tutorials/vim_tutorial.html
[77] http://yehudakatz.com/2010/07/29/everyone-who-tried-to-convince-me-to-use-vim-was-wrong/
[78] http://vim.wikia.com/wiki/Vim_Tips_Wiki
[79] http://Vimcasts.org/
[80] http://stackoverflow.com/questions/2573/vim-tutorials
[81] http://vim-adventures.com/
[82] http://www.youtube.com/watch?v=szrsfeyLzyg

Vim: *"There is a plugin for that"*. Janus[83] is a comprehensive distribution with plugins for Vim (and GVim, MacVim).

The documentation of Janus explains all the nice features Janus integrates. There is a plugin[84] which checks that your code syntax is correct, an integration of snippets[85], simplified way[86] to browse and load files etc.

[83] https://github.com/carlhuda/janus

[84] https://github.com/scrooloose/syntastic/

[85] http://Vimeo.com/3535418

[86] https://wincent.com/products/command-t

File Sharing - Share the Stuff!

Do I Need a Filesharing System?

Yes you do.
You won't send files by e-mail or skype anymore and believe us it will save you a lot of trouble. No more files you forget to download, no more setting up your own download directories, no more searching for email attachments. Everything is in one place. Updating is done in the background, you get notified everytime something changes in your folder.

Use Dropbox

We chose Dropbox because it's very simple to use, you can easily invite people to collaborate, it works on most devices and operating systems and it's cheap. Dropbox is like a regular folder, that automatically syncs across your and your collaborators' different devices meaning that when you put a file in your Dropbox, it will appear on your other devices and on your collaborator's devices as well. It just works!

Why Not Use Github?

Github is great for content that needs collaborative editing. But when you just want to share files that are unlikely to change or when you don't really care about the history of changes and just want to use the last version every time it's updated (PDFs, PPTs, etc.) it is more convenient to use Dropbox. No need for commands and manual syncing in this case. Non tech-savvy clients and employees will use it without any training as it is so simple.

Setting up Dropbox

Simply go to the Dropbox website[87] and click 'download'. Once you are done downloading and installing, you'll have a Dropbox folder on your system that you can organize like a normal folder. You can also setup a collaborative folder on their website.

[87] https://www.dropbox.com/

Bookmarks and Notes - Don't Lose Track of Good Content

Why Do You Need to Bookmark?

Whether you are browsing the web looking for resources or help, following the latest news on Twitter, aggregators or RSS feeds, you will stumble on a multitude of interesting things that you might want to go back to later. You cannot keep track of everything by yourself unless you have eidetic memory[88]. This is where bookmarks come in.

Sharing Content in a Team

Working in a team means sharing information and resources. You can still do it by email or by chat but you will struggle to find it later. What you need is a collaborative bookmarking place that is searchable.

Working with Evernote

We chose Evernote to fulfill our needs. It is much more than a bookmarking tool. With Evernote you can save **for offline use** personal notes, files, links and organize them in notebooks. It also lets you add tags to your bookmarks so that you can retrieve them later.

It's more powerful than Delicious[89] especially because of the offline mode and the clipping feature, which lets you save what's really important in a webpage and only this. You get to pass on the ads, asides, menus, and other visual clutter.

In addition to the web app it offers desktop and mobile apps. Bookmark on your desktop, read it while you are on the subway, or the opposite. Evernote also makes sharing dead simple. It's basically Delicious on steroids.

Setting up Evernote

- Register on Evernote[90]. All you need to know to master Evernote can be found on this page[91]
- Install the browser plugin to add new notes to Evernote

[88] http://en.wikipedia.org/wiki/Eidetic_memory
[89] https://delicious.com/
[90] http://www.evernote.com
[91] https://support.evernote.com/ics/support/KBSplash.asp

- Install the desktop and the mobile version
- Create a shared notebook with your team (accessible with the Premium account)

Real Life Example

Each one of us clips articles or slide decks that will probably be useful at some point. They are mostly about technical topics, user experience, design, or insightful views on technology trends. Thanks to the tags and full text search it is easy for anyone of us to then access these notes.

Tracking Expenses - Stay Good Friends!

Why an Expense Tool?

You will incur a number of common expenses like github, the server, food and so on. Tracking them can be painful, but shouldn't be. To avoid keeping pieces of papers or a detailed spreadsheet we use Tricount[92], a simple and lightweight expense tool.

Organize Your Expenses

With Tricount you don't need to devise a complicated formula to compute everyone's balance, you don't have to fill in tons of information and you can add new expenses in a few seconds.

You can easily organize group expenses, specifying who paid for what, who are the beneficiaries, whether the distribution is equal among them or not, and so on.

Record every expense in your Tricount account and at any moment you can visualize how much money everybody spent and who owes how much to whom.

Share Them Easily

Tricount makes it simple to share expenses with other people. No registration is needed, simply a link.

It comes with a web site and mobile apps for Android and iPhone[93]. Go for it and try it, it will drastically simplify keeping track of expenses with your teammates.

[92] http://tricount.com

[93] http://itunes.apple.com/app/tricount/id349866256

1+1 > 2

Wait what? Yeah you heard me. We thought merely listing good tools was not enough. What's more interesting is how to make them work together so that you use the least amount of tools and yet benefit from an optimal workflow.

It's like what Apple does with their products: it all works better if you have an iPhone, an iPad, and a MacBook rather than an Android phone, an iPad and a Windows-based laptop.

Most of these tools can serve multiple use cases when they are combined with one another, and that means you need less tools, spend less time learning new tools and more time creating your product.

HipChat and Github

Hooks

This is super important, maybe the most important synergy of all. You can set up Github hooks to HipChat in order to be notified in your chatbox of every change to your repository. It takes 7 clicks and a few keystrokes to set up.
We recommend creating one room per repository. This way you get alerted in a cleanly separated way, and you can follow activity seamlessly. There is no need to check the Github activity feed anymore, or ask if someone already pushed the change you were waiting for. It's there for you to see in your chatbox.

Hubot

The ultimate add-on for your HipChat chat room is Hubot[94]. Hubot is a bot that lives in your chat room that you can invoke to run custom actions. It has been created by the fine folks at Github as part of their quest to automate everything possible. Have a look at some of the custom scripts[95] available. You can of course create your own scripts.

[94] http://hubot.github.com/
[95] http://hubot-script-catalog.herokuapp.com/

Bash and Vim

The fact that Vim can be opened in a terminal window is a big deal. It means that you don't need to always open a new window for quick reviews or modifications of files, which will save you some time and hassle if you know Vim reasonably well.

Furthermore, since you'll be interacting with your server with the Terminal, if you want to edit any file that's remote you won't have a choice but to use a non graphic editor, such as nano, vim or emacs. Sorry, no more Textmate or Sublime Text on remote servers. Therefore getting used to Vim will prove very valuable for when you need to get some work done remotely.

Did we mention that Bash had a *vi* mode[96]?

Bash and Git

Git's primary user interface is the command-line. But you can make it better by configuring your terminal with Git auto-completion and all sorts of great tweaks. For instance, you should add your current branch to the prompt to prevent mistakes and work faster.

Here's how to set it up[97].

Vim and Git

Fugitive[98] is[99] a[100] great[101] plugin[102] for Vim that will enable you to use a lot of Git from inside Vim, which is awesome for diffs, resolving conflicts and other things.

Bridge'em all

Most of the tools listed in this guide provide an API. You can develop whatever seems appropriate to you to create synergies using the Github [103], Trello [104], HipChat [105], Evernote[106] or Dropbox[107] APIs.

[96] http://www.catonmat.net/blog/bash-vi-editing-mode-cheat-sheet/
[97] http://nuclearsquid.com/writings/git-tricks-tips-workflows/
[98] https://github.com/tpope/vim-fugitive
[99] http://vimcasts.org/episodes/fugitive-vim---a-complement-to-command-line-git/
[100] http://vimcasts.org/episodes/fugitive-vim-working-with-the-git-index/
[101] http://vimcasts.org/episodes/fugitive-vim-resolving-merge-conflicts-with-vimdiff/
[102] http://vimcasts.org/episodes/fugitive-vim-exploring-the-history-of-a-git-repository/
[103] http://developer.github.com/v3/
[104] https://trello.com/board/trello-public-api/4ed7e27fe6abb2517a21383d
[105] https://www.hipchat.com/docs/api
[106] http://dev.evernote.com/documentation/cloud/
[107] https://www.dropbox.com/developers/reference/api

Start developing you own tools[108]!

In fact, ifttt[109] has already done some bridging work, go check it out!

[108] http://needforair.com/blog/2012/03/05/stop-procrastinating-start-scripting/

[109] http://ifttt.com

Who Are We?

Need for Air[110] is composed of Louis Chatriot, Stanislas Marion and Charles Miglietti. Sharing the same passion for the internet and entrepreneurship, we quit our day jobs at the same time to launch a company together. We are working on making good content more easily discoverable through better curation. Previously we learned to work together on our first (and only so far) freelance mission for, Captain Dash[111], Laurence Parisot[112] and the Medef[113].

Louis Chatriot

I'm a long-time geek who's always wanted to understand how things work, especially the fields of math, physics, computer science and economy.

After 6 months as a software engineer at Robinsons Participations and about 2 years as a management consultant for Bain & Company, I decided to launch my own company with Charles and Stan.

I graduated from Ecole Polytechnique (majored in CS) and have a MS in Management Science and Engineering from Stanford University.

Stanislas Marion

I've been meaning to start a company for a long time. I quit my job without a firm idea nor a cofounder. It didn't take long to find both. In the past I successfully played online poker at middle to high stakes, which taught me a lot about risk-taking, long-term focus and short term swings, adapting and the value of money.

I worked for about 2 years at a Parisian startup[114], where I learned a ton about small team dynamics and problems. I graduated from Ecole Centrale Paris and have an MS in Management Science and Engineering from Stanford University.

Charles Miglietti

I do what I love and I'm always on the lookout for a good challenge. Being passionate about computer science and math, as well as innovation,I became more and more interested in entrepreneurship.

[110] https://twitter.com/#!/NeedForAir
[111] http://captaindash.com
[112] https://twitter.com/#!/laurenceparisot
[113] http://www.medef.com/medef-corporate.html
[114] http://yseop.com

After graduating from Ecole Polytechnique, I spent a year as an embedded systems engineer at Withings[115], one of the most promising French startups at the moment. Inspired by this experience, I decided to quit to start my own company.

[115]http://withings.com