Dynamo + GitHub = DynaHub: Adopting a New Innovative Repository for Dynamo Content
Andrea Tassera
Willow

Learning Objectives

- Discover the value of adopting version control in computational design;
- Learn how to set up your version control strategy with DynaHub: how to create and manage your GitHub account, and load your Dynamo graphs;
- Learn how to commit and push your changes to the Dynamo graphs: why, when, and how;
- Learn how to manage and distribute Dynamo automations and packages across your organization, and how to collaborate the easy way.

Description

When adopted throughout a company, people can be intimidated by visual programming, Dynamo can feel confusing, and collaboration can be difficult. Therefore, managing Dynamo libraries is labour intensive. GitHub is widely used in software engineering for facilitating collaboration and consistency of code over time. However, it has never been applied in computational design. DynaHub—an open-source Dynamo extension that connects Dynamo to GitHub—bridges a gap by letting you push and pull automations and packages from a centralized and accessible repository. Any user or organization, through their GitHub account, can use the Dynamo scripts stored in their own repository, knowing that they are using the most recent and validated version. By adopting this strategy at Willow, we experienced an increase in productivity, confidence, and usage.

Speaker(s)

Andrea is an Architect and Engineer with extensive experience in computational design and in software development. He started experimenting with Grasshopper during university, fascinated by complex geometry and the architectural functions of a building. Previously serving as the Computational Design Manager at Willow, his role has now expanded to Technology Specialist. In addition to his design-focused role, he now also develops and leads the creation of software applications and tools which benefit both the business and customers. During his work with Dynamo implementation, he further developed his skillsets in .NET programming and re-focused his attention into solving problems related to data management, validation and model auditing with the use of the Revit and Dynamo APIs. Andrea is also member of the committee of the Australian Dynamo User Group, frequent presenter of the same and speaker at the yearly Computational Design Workshop Conference held in Sydney.
Introduction

The amazing thing about Dynamo is that, like most visual programming tools, it brings architects, engineers and non-programmers in general closer to the world of automation. This brings great benefit to the industry by increasing efficiency and consistency of workflows, freeing time, saving money and expanding the spectrum of possibilities in the industry.

Despite the great benefits it brings and the ability to get more people involved with automation, Dynamo still has the stigma of being a “geeky thing” and too complex for the masses. I tend to agree with this point of view for two reasons:
1) Programming, even if visual, is hard to pick up and requires time and commitment;  
2) Dynamo has imperfections with its user experience and some operations can become a bit frustrating.

While working at Willow and managing the use of such tool(s), I started recognising recurring pain points in the use of Dynamo for my colleagues, mostly in the use of scripts that I was creating for them to use.

Therefore, learning from the world of software development and how they solve these problems, I came up with the idea of DynaHub, an extension for Dynamo that aims to seamlessly connect the visual programming tool to an online repository (GitHub in this case).

What problem does DynaHub want to solve
The question that I’ve been hearing the most when people approached me wasn’t “how do I use this script?”, as you would expect, but actually “where do I find this script?”. No matter how hard I tried to establish conventions, central storing locations for our graphs, communication and re-iteration of the message. People constantly asked me where to find the scripts. Most of the time, I blame laziness.

Second place goes to “Dynamo doesn’t work. I ran the script, nothing happened and now the graph is all colourfull!”. You are guessing right: after several minutes of analysing the problem, that person just didn’t have the right packages. Now this is a serious problem with Dynamo. If you develop and use your own graphs everything’s going to be ok, but if you want/need to pass it on to someone else to use, the other person will need to have the same packages (oftentimes in the same version) in order for the graph to work.

Another thing that will happen is someone that will open the original file in its location, overwrite the original graph and save. You might lose information and time, or you could even damage the model in some case. Rolling back to a previous correct version of the graph could be hard in that case.

Finally, collaboration is quite hard with Dynamo. If you have multiple computational designers in the team and they are working together on a workflow, they will have to constantly exchange the working files, create copies and waste a lot of time.
Are these problems worth solving?
As a general consideration, software development wouldn’t exist as we know it today without the possibility to store code in a central cloud-based location, accessible to all the team members everywhere in the world, it acts like a ledger for changes, ensuring consistency over time, thus facilitating collaboration.

Visual programming should evolve and learn more from its older brother (software development), as there are proven benefits. After all, you won’t see Google storing their code on a laptop and sending around the text files whenever the code needs to be changed. Why isn’t the intellectual property of that kind of information considered valuable in the AEC industry too?

Moreover, there are other immediately tangible benefits. By removing all the mentioned constraints, people will use Dynamo more, saving time and money for the company. By adopting this strategy at Willow, we have experienced an increase in productivity, saving an average of 30 minutes per use case of Dynamo. This, multiplied by the number of use cases throughout the day and the number of Revit users within the company, produced large amount of savings.

Making Dynamo less intimidating will result in higher uptake of the tool. Therefore, further savings in time and cost to a business. Moreover, more people picking up Dynamo to automate their tasks will mean that those tasks are going to be performed by the computer, minimising human error and maximising consistency and best practices. This, in the long run, will add to the efficiency and savings.
Version Control in Computational Design

The distribution system discussed above is what, in the world of software development, is called **version control**, or **source control**.

What is version control?
Version control is the component of software management that is dedicated to handling the changes to documents (including of course codebase) and tracking its full history. It could be described as a ledger, where changes are packaged and stored in a timeline with an identifier, a timestamp and other useful information. These “stored slots” are called **revisions** and because they can be identified and placed in time, they can also be compared to each other, restored as current and sometimes merged.

To sum up, version control is a system that records changes to a file or set of files over time, so that you can recall specific versions later in time.

This strategy speeds up the process of creating software by **improving collaboration** between programmers, increases data integrity by allowing to roll back to previous working versions of the codebase and enables non-linear workflows thanks to its tree structure, through forks and branches.

There can be several branches (versions) of the code in a version control system, according to the number of collaborators and features being developed. The branches maintain individuality as the changes made to the code remain in a specific branch.

Version control also allows developers to combine code changes when required, view the history of the changes and go back to previous (working) versions of the code if needed.
What is Git?
A version control system (VCS) is a platform / software application able to manage this process.

Probably the most famous and utilised VCS around, is **Git**. Git is a **distributed** VCS. This means that the version control, including the whole codebase, its history, revisions, etc. are mirrored on every participant's computer. This has the benefits of speeding up most operations, allowing programmers to work offline and making the system more secure not relying on a single location.

Git is operated by its own set of commands usually typed through the **command line**. These commands allow you to create or clone existing repos, check the status of it, commit changes, create and merge branches, etc.

It’s always recommended to become familiar with the command line when dealing with Git. First because the commands are extremely simple and not many in number, but also because it brings a deeper understanding of the technology and how it is operated.

Although, the purpose of this class is not teaching you Git per se, so we’ll be using a more friendly graphical application in order to interact with Git. That will abstract a lot of the commands in something more familiar. My suggestion though is that at least once, you experience the thrill of doing things from the command line.
What is GitHub?

Finally, we have GitHub. GitHub is one of the many web services that host Git online. It offers all the version control and source code management features of Git, but it’s ubiquitously reachable on a website ([https://github.com](https://github.com)) and adds its own functionalities. This includes access control such as levels of permission to the project, and collaboration features such as bug tracking, communication with the users, feature request, wikis, etc.

GitHub is the most common and most utilised ([https://venturebeat.com/2018/11/08/github-passes-100-million-repositories/](https://venturebeat.com/2018/11/08/github-passes-100-million-repositories/)) online system for version control and the largest host for source code in the world. For this reason, it has been chosen for DynaHub.

In GitHub you would create an account as a user (that can be then grouped by organisation/company) and then the user creates repositories. Repos, as they are often called, can be seen as folders that will contain your project files. The repo contains everything that refers to your project: text files with the code, readme files, images and so on, together with the history, the branches, etc.

Repos can be public (in that case you have open source projects, such as DynaHub) or private so that only you and your teammates can see it and work on it.
What is a Dynamo Extension?

Version control is a concept applied not only to big and complex software projects. This habit of checking in your code in a VCS is so rooted in software developers that almost every IDE (Integrated Development Environment), such as Microsoft’s Visual Studio for example, will have an integrated section or at least a plug-in that will seamlessly manage the connection with your favourite VC platform.

Conceptually, it’s easy to compare Visual Studio and Dynamo (or more generally software development and visual programming), being both applications where you write your programs in. Although the outcome would look rather different most of the times, the idea is the same.

However, the process to interact with GitHub is very different: it tends to be quite clear in the world of software, but almost unexplored in visual programming.

If a new developer joined a software team, he/she would easily connect to the company’s GitHub account, download the current code and start working on it straight away, being able to seamlessly collaborate with the other team members, all without leaving the IDE.

Doing the same thing in Dynamo would mean leaving Dynamo in the first place to find workarounds, which could involve the command line, direct downloads, etc. All this is reliant on the firm implementing some sort of version control strategy for Dynamo and having a GitHub account in the first place (which I am yet to see in my experience).

Luckily enough, not long ago, the Dynamo team released a set of APIs that help you create Extensions for Dynamo. Extensions allow the programmer to “hack” Dynamo itself and add functionalities to it, like plug-ins.
Dynamo creates programs/algorithms that automate Revit actions within its canvas, while Dynamo Extensions allow you to create programs that give the Dynamo ecosystem itself new functionalities. These APIs aren’t the easiest to use due to the lack of documentation, but at least they’re public.

Difference between a Dynamo graph/algorithm and an Extension.

Read more here [https://developer.dynamobim.org/03-Development-Options/3-6-extensions.html](https://developer.dynamobim.org/03-Development-Options/3-6-extensions.html)

This new possibility allowed the creation of DynaHub, an extension that allows easier communication between Dynamo and GitHub, without the need to leave Dynamo. This plug-in is still in development and will keep adding new features. It is also open source, so you can just visit [https://github.com/Dre-Tas/DynaHub](https://github.com/Dre-Tas/DynaHub) and fork the repo to contribute and help it expand. Any help is more than appreciated and there will be a section at the end on how to get started.
Version Control Strategy for Dynamo

In this chapter we’ll have a look at how to easily implement a consistent strategy to leverage the power of GitHub within Dynamo, not only for keeping track of the changes in your graphs, but also to have an easy way to deploy your automations across organisations.

We’ll start from the very beginning.

How to create a (free) GitHub account
Creating an account on GitHub is straightforward and not dissimilar to creating an account for any other web service you normally use, such as social media or news.

You just need to go to https://github.com and you should see something similar to the screenshot below. If not, click on Sign up to get here.

Built for developers

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 40 million developers.

GitHub’s landing page.

Simply fill in the required information, such as username, email and password.

The next step will require you to select the type of subscription you want.
In this class we will only look at the free version of GitHub, which will allow you to create public repositories, but also private ones, which is all we need for today’s purpose. If you intend to use DynaHub within a large company, with many Dynamo users and many people collaborating on the same repo, you might consider subscribing to the Pro (paid) version, but that’s outside of today’s scope.

You don’t need to tick any of the checkboxes, as per the screenshot above.

Finally, the last step prompts you to tailor your experience by informing GitHub what are you going to use your account for. Just follow the simple instructions. As this step is not strictly required for the following tutorial, you can skip this step for now.
Learn Git and GitHub without any code!

Using the Hello World guide, you’ll start a branch, write comments, and open a pull request.

Read the guide

Welcome to GitHub

Welcome to GitHub—where millions of developers work together on software. Ready to get started? Let’s learn how this all works by building and publishing your first GitHub Pages website!

Repositories

Right now, we’re in your first GitHub repository. A repository is like a folder or storage space for your project. Your project’s repository contains all its files such as code, documentation, images, and more. It also tracks every change that you—or your collaborators—make to each file, so you can always go back to previous versions of your project if you make any mistakes.

This repository contains three important files: The HTML code for your first website on GitHub, the CSS stylesheet that decorates your website with colors and fonts, and the README file. It also contains an image folder, with one image file.

Automatically created first repo when you create an account.
How to create a new repository

As you can see from the previous screenshot, once you have successfully created and set up your account, GitHub will automatically create your very first repo. This will mostly contain instructions on how to use GitHub and other guides.

Although, a time will come when you want to create a new repository, where you can host your super-cool new project (or Dynamo stuff in this case).

Where to create a new repository.

Looking at the screenshot above, there are two equally simple ways of creating a new repo. Just go to your profile's main page and:

1) In the top-right corner of the page, close to your small profile image, click on the + button and select New repository from the dropdown list;

2) In the top bar, just below the dark grey bar at the very top, go to the Repositories section (where you’ll see a list of all your repos and the ones you forked) and click on the green button that says New.

This will take you to the Create a new repository page, that will prompt you to insert all the basic information needed for creating the new container.
Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner

Repository name *

Owner

Repository name

Create a new repository page.

The mandatory info you'll be required to insert are, the name, and if you want the repository to be public or private.

A **public repository** can be accessed by everyone (and not just the people with a GitHub account) and everyone can see the code that composes your project, while a **private repository** can only be accessed by yourself and the people that you add as collaborators to the project.

For this reason, in this class we will create a private repository, simulating the scenario of a company that stores automations that might contain sensitive data that cannot be shared externally. Also, a private repo will allow me to demonstrate that DynaHub is able to access both public and private repos, giving you greater flexibility.
Of course, you’re free (and encouraged) to create a public repo where Dynamo graphs are shared and contribute to the community, similarly to what we’re doing at the Dynamo Sydney User Group (https://github.com/dynamo-user-group).

Additionally, there is an option to:
- write a short description of the repo to tell what is the project about;
- automatically create a README file that gives in depth information on the project and would be the first file in the repo (make sure you create one now);
- use a predefined .gitignore setting to avoid uploading certain file types to GitHub (quite useful, but we’ll bypass it in this case);
- add a license, mostly useful for public projects, retaining some rights on your creation.

You’ll then simply hit the Create repository green button to prompt GitHub to work its magic and create the repo for you.

What you’ll see should be similar to the image below (without the red numbers of course).
Let’s have quick a look at some of the fundamental parts of the of the page and where to find some basic information that you will need from time to time:

1) At the very top you’ll see the name of the repository, composed from the username and the name that was given to the repo at creation. This reflects the URL, that will be something like github.com/username/repo-name. Here you will also see a watermark telling you if the repo is private;

2) A very quick summary of how many times users committed code and how many branches are active. More on the meaning of this in the next chapters;

3) Which branch you are looking at. Different branches will be likely to have different content, so you’ll always need to be aware of which version of the project you are looking at;

4) The three grey buttons let you create, upload and find files in the branch. Of course, you wouldn’t just create or edit your programs from here, as the capabilities are limited. The green button will help you download the files locally on your machine. As an example, that’s where you’d go if you wanted to open the project with Visual Studio or GitHub Desktop;

5) This rectangle will show you all the files and folders that are in the repo. The blueish top part shows the latest commit, its name, when it has been done and who has done it. Below it you will see a list of the files, which one was the latest commit to edit it and when did that happen.

You can also see a very simple example of a readme file just below the list of the files. In any folder of the repository, if you put a file named readme.md, it will be automatically shown as a formatted and pretty text that can contain links, lists, images and more. (To create a nice readme file you can have a look here https://help.github.com/en/articles/basic-writing-and-formatting-syntax)

We will go into more detail in next chapters as we use those parts of the repo.
Easy interaction through GitHub Desktop

As previously mentioned, usually the way to start learning how to interact with GitHub is through the command line, if you follow online tutorials. I believe it is good practice and will help you understanding better how GitHub works. Although, in this class, which is not strictly about GitHub, we’ll cut a few corners. The command line could be quite intimidating to many and this is the opposite of DynaHub’s aim.

Therefore, we’ll interact with GitHub using the GitHub Desktop application, which gives a nicer and more intuitive interface to this communication.

Passing the control of the repo to the desktop application is simple and you just need to click on the Clone or download green button and select the Open in desktop option.
If you don’t have GitHub Desktop, you can click on the link that appears in the balloon, that will redirect you to GitHub's download page, or from here [https://desktop.github.com/](https://desktop.github.com/).

This will be the opening screen that you will see on GitHub Desktop:

![GitHub Desktop app.](image)

Now, specify where the online repo should be copied locally (below Local path). This step will copy what’s on GitHub to your machine (to the location you are here specifying), while creating in the background a “link” to what’s online, meaning the local folder and the online folder/repo will be able to recognise each other and keep in sync. This is the meaning of GitHub as the online host of Git. This works because you still have the Git repo locally on your machine and it syncs with GH online.
How to structure the repo for the best DynaHub results

DynaHub doesn't require any specific folder structure, apart from one little thing. (But of course, you wouldn’t want your Dynamo dedicated repo to be complete mess).

The one little thing that is necessary for you to do if you want to share your packages with your colleagues, is to create one folder named packages. Be aware that this name will be case sensitive.

We’ll discuss this in far more depth in one of the later chapters.

Example of what a repo structure should look like.

Above you can see an example of a simple repo structure.

The repo contains a readme file that can give some information to your user and could contain, for example, a list of all the automations uploaded to GitHub and what they do. (This file will not be detected by DynaHub as it only looks for zipped folders and Dynamo graphs)

It also contains the required packages folder for the distribution of the necessary additional packages.
Finally, all the other folders are just a way to organise your dynamo graphs (.dyn files) tidily. You would probably store project-specific graphs in a project-specific folder, so that the user would find it more quickly. You might also want to put graphs that are used on multiple projects in a container such as the GeneralWorkflows one in the image. I personally also like to have one folder where I store the templates to help me start new graphs. Having a sort of work in progress folder (testing in the image above) where I can put the graphs I'm working on is also handy. This last one could also be quite useful in case of collaboration with other Dynamo power-users.

Furthermore, be aware that you can nest the folders as much as you wish (folders and sub-folders) as DynaHub will recreate the same structure when showing you the available graphs.
GitHub management basics for DynaHub

In this chapter we’ll get practical on GitHub. We’ll try to keep it as simple as possible, showing you just what is needed in order to successfully use DynaHub and be able to deploy you graphs in a smart way. Keep in mind that there is so much more you can do than what we’ll cover here, so I suggest you also have a look at it on your own.

At this point you should have created your GitHub account and the repository that will host your Dynamo files. But the repo is still empty. In the next paragraphs we’ll see how to:

- upload files to your repo
- understanding the difference between “committing” some files and “pushing” them
- how to update your local copy of the repo and
- a first look at how teams can collaborate on the same files.

In the next paragraphs we’ll be using the terms Git and GitHub almost interchangeably, as GitHub could be simplified to *Git hosted on the web*, but please be mindful of the fact that they are two different things.

All the commands that we’ll see below are really Git commands, but of course they are reflected in GitHub.

Add files to your Dynamo repo = commit

“Committing” files to GitHub is probably one of most confusing topics, at least in my experience. Committing doesn’t mean that you are uploading files to GitHub. You could see the *commit* command like telling GitHub (or really Git) to *keep an eye on this file*. So, this command is used in connection with your *local repository*. To better understand where this comes from, think that Git was born for software development, where many files are being written. Not all of them contain code that is useful to the project. They might contain setups, notes and files that you might not want to share. Hence, you probably don’t want to keep track of the history of all those files.

The *commit* command usually requires one parameter, which is the *message*. To help you remember what changes you have done in any commit you have done, you are required to write a short description. By “usually” I mean that, if you are committing a single file, the GitHub Desktop application will write a simple message for you based on the file that is being committed. The message is still required, but it is already taken care of.
Now let’s see how to do it practically. First, open the GitHub Desktop application.

On the top left corner, where you see Current repository, click on the triangle to expand the repos section. Here you will see a list of all the repositories managed by the application.

See the list of repos in your GitHub.

Having cloned your Dynamo repo from GitHub in the previous chapter, you should already see it in the list.
If you cannot see the repo in your app, you can always add it. To do so, click on Add to expand the dropdown box and then click on Clone repository. This will open a dialog box where you can select which repo you want to clone locally from a list of all the repos associated to your account.

Once you select the repo you want to work on, you will see something like the image below:
On the top left corner, you will see information about the repository you are currently in, such as the name and, in this case, the fact that it is private (expressed by the lock symbol on the left of the name).

On its right, you can open the menu to deal with branches, create new ones, switch between them, create pull requests, etc. More on this later in the chapter.

On the right of the branches button, you have the Fetch origin button that will, if needed, update your local version grabbing what’s online on GitHub. This is particularly useful when you are collaborating with someone else.

In the main part of the screen, on the left, you will see a summary of the files that have changed in the repo, with the ability to see where and what has changed. At the bottom, where you now see Summary (required) you will have to write the message that accompanies the commit command and, optionally, a description of the commit.

Lastly the Commit to branch (which is the master branch now) that is not active right now as there’s nothing to commit.

Looking at the top part of the left side of the main screen, you can switch to the History view, where you will see the history of your commits, with their messages, date and time and other info.

On the right-hand side of the main screen, you will be able to go into a more detailed view of the changes that happened to the files you are committing.

When there are no changes to show, you have a few possible actions you can perform to start committing files. The most useful one for us right now, is Show in Explorer that will open the local repository / folder in your Windows Explorer (aka a normal folder browser). Opening the repo as a folder will give us an easy way to add or delete files from our repository.

Click on the Show in Explorer button. You should see a Windows folder appearing, pointing to where the repo has been saved locally on your machine and it should now just contain the README.md file, if you created one.

In case the Show in Explorer button wasn’t there, you can always open the folder by going to Repository in the top-most toolbar and click on Show in Explorer, or use the shortcut Ctrl+Shift+F.

Now, let’s try to add some files to our repo. Go to the folder containing the Lab’s .dyn files, select the first two to drag and drop them to the local repo folder that just popped up. Easy!
Two files have been added to the repository.

The two files that you dropped in the repo folder are now automatically detected by GitHub Desktop as a change made to the repository and shown in the app. On the left hand-side you’ll see the two files are being added (green plus sign on the right of the name) and on the right all the code that is being modified. In this case all the code is in green, as it is all being added.

So, we can see that dropping the files in the folder is automatically recognised by the desktop app as an add command. For this reason, we’re not spending time on that, as it is all being added.

To commit the files to our local repo, all we need to do is write a summary message, that could be for example “uploaded first two files”, or something more meaningful if this was a real world scenario. To complete the task, hit the Commit to master button, that will be enabled once we write the message.
The app goes back to a very similar look to what it was at the beginning.

GitHub Desktop’s screen after committing.

At the bottom left, some text will appear showing the last commit made, giving also the option to Undo in case you made a mistake. On the right, the option to push to the web appears. We’ll discuss this in the next paragraph.

Committing (and pushing in the next chapter) are not functionalities of DynaHub yet. That’s why we are doing it through GitHub Desktop. (Although, it is my intention to add them. I just need to understand what’s the best way to do it. Every suggestion is welcome)
Upload files to your Dynamo repo = push
Having defined the meaning of the commit command, it should be easier to define the second fundamental command, push.

Pushing is more straightforward in what it does. It simply takes all the commits that have been done and “pushes” them to the online repository, updating it.

If you look at the top part of the diagram below, you can see that push takes over just after commit. Although, this isn’t an automated process and you always have to send the push command yourself.

Git Data Transport Commands

Be also aware that one push can include multiple commits, as we are going to see now.
From the folder with the Lab’s material, drag and drop the last file to the repo’s folder and commit. Notice that the message has already been filled out for you.

Now, just click on the *Push origin* button, either the blue one or the one at the top, to see the changes reflected on GitHub online.

If you now go to the webpage of your repo (something like: github.com/username/repo-name) you should see the files appearing. Those files are on the internet now, which means, with the right tools (DynaHub, anyone?) and credentials, you can almost access it from anywhere. Pretty powerful!
Update files from your GitHub repo = fetch + pull
Sometimes it will happen that your local repo will go out of sync with the online repo. This means that what you see in the local folder of the repo is different from what is stored online on GitHub.

This scenario could happen if, for example, you uploaded or created some files directly from the web repo, or if you are collaborating with someone else that is also working on the repo.

In order to recreate the scenario, I’ll create a random (useless) .txt file from the online GitHub platform, simply by clicking Create new file and populating the required info. Finally, I’ll click on Commit new file at the bottom.
You can now see that the text file is in the list of files, but if you check your local folder, it’s not there.

The file just created is not in the local repo.

This is what it means for the local repo to go out of sync. Something happened online, but our local repo is just a normal folder, and it’s not aware of what happens in the remote repo.

But fear not. On GitHub Desktop, at the top, click on *Fetch*. The app will reach the remote repo and see if there are differences with what you have locally. If differences are found, like in this case, the option to *Pull origin* (pull the differences from the origin, the remote repo) will appear.

Differences are found between the local and online repo.

Now, click the button to see the file created online appearing in your local folder.
Create your own testing ground = branch

Warning: this is something that I haven’t tested myself, so it will merely be indicative and informative. If you try workflows that include the following, I’ll be grateful if you reach out and let me know what you did and how.

When collaborating within teams, everyone will have their own version of the repo locally and push to the origin (the remote, shared repo) from time to time. Collaborators will be able to work on the same file at the same time and then merging the two versions together and fixing the conflicts, but this goes beyond of goal the of this Lab.

It generally is good and common practice when collaborating to divide the repository into compartments so that can be assigned to different people. In this way you can develop different scopes or features of the program in those compartments and collaboration will be easier.

The compartment is called a branch in GitHub.

How to create a new branch.

To create a new branch, click on the dropdown button Branch: master and simply write the name of the branch you want to create, then click Create branch: branch-name. When you’ll click the Branch button in the future, you’ll see a list of all the branches you created.

We have already seen above where you create a new branch in GitHub Desktop.
Bring your edits to the main code = pull requests and merging
When you think your job is done in the branch that you created, you will want to merge the changes you’ve made to the master branch (the main branch).

Let’s create a new file in the new branch so that the two branches (the master branch and the new one) will not be identical.

<table>
<thead>
<tr>
<th>MyCoolDynamoRepo</th>
<th>branch-specific-file.txt</th>
<th>Cancel</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td></td>
</tr>
</tbody>
</table>

Create a new file in the new branch.
On the right of the Branch: branch-name button, click on New pull request.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

Able to merge. These branches can be automatically merged.

Create branch-specific-file.txt
- Write
- Preview
- Leave a comment

Open a new pull request screen.

You are now opening a new pull request (or PR) and GitHub will give you some info.

The one you will be looking for the most is in the red rectangle at the top. It says that there are no conflicts with the existing code so you can merge automatically.

Also, immediately below, you will have to write a title for the request, similarly to when you commit. Additionally, you can write a brief description of the PR so that it’s going to be easier for the person examining it to understand what’s going on.

At the bottom of the page you can also see the changes that have been made and what is about to be merged.
When you click on *Create pull request* in green, you will be brought to another screen where, in this case, you just click the green button saying *Merge pull request* and then *Confirm merge*. At the very end, you can decide if you want to delete or keep the branch you just merged.

Be aware that it was this easy because there were no conflicts and the code could be merged automatically. In other cases, you’ll have to go through the code and fix the conflicts.

In traditional software development teams, at this point, a senior engineer or a technical lead will have a look at your pull request, check the code you are submitting, compare it with what already exist, run some tests and, in case everything is right, merge it to the master branch.

Now, how does this look in Dynamo? This is an amazing question that makes me dream of wonderful worlds that don’t exist yet. For sure this is something that needs to be inside of DynaHub, including the creation of the branch, the pull request, the comparison, etc.

I haven’t understood what’s the best way to do it yet though. As always, feedback and suggestions are welcome.

**“Stealing” from others = fork**

The whole idea of DynaHub is also promoting open source and sharing material with each other whenever possible.

Someone might release publicly the repository he/she created with his/her Dynamo workflows, similarly to what we’re doing with the Dynamo User Group in Australia.

In that case, you can fork the public repository and see that a copy of that repo appears in your profile page, as if it were yours.

From this moment on, you can use the repo with DynaHub, and you can even change the files that are in it without messing up the original repo.

After you have made your edits, if you think you have improved the original repository and the original owner might be interested in it, you can also create a pull request to merge to the original repo. And in this way you become an open source contributor!
DynaHub operations and how to use it

Here we’ll have a look at DynaHub itself, understanding its usage. You’ll learn how to:
- install DynaHub;
- login to your GitHub account without leaving Dynamo;
- view the content associated to your GitHub account in Dynamo;
- open online-hosted graphs;
- how to update your packages to the official ones shared by your company.

We’ll also have a look at a possible strategy for an organisation to store and deploy automations, while hopefully making collaboration easier.

For ease of presentation, I’ll be using the Dynamo Sandbox to demonstrate DynaHub’s operations. Be aware the exact same actions can be carried on with any version of Dynamo inside of any version of Revit. I suppose it should also work for any other version of Dynamo, such as Dynamo for Civil3d, but this hasn’t been tested.

DynaHub installation

As mentioned at the beginning of this document, DynaHub is an Extension (or a plugin) for Dynamo. The process to install extensions in Dynamo is the same as installing packages. Extensions are loaded in the Dynamo Package Manager in the same way as packages. They just contain different files and structures.

Simply go to the package manager from Packages > Search for a Package… and type DynaHub (or dynahub, as it is not case sensitive) to find the package.
Download the latest version by either selecting it from the list or clicking the big down-facing arrow in the circle.

You will see that the extension has been successfully installed because DynaHub will appear in the top toolbar, together with the other custom extensions you may have previously installed.

![Successful installation of DynaHub.](image)

Click on the DynaHub tool, it will show the functionalities of the tool that have been developed so far.

**Log into DynaHub**

DynaHub doesn't have a registration for itself as it just facilitates the communication between Dynamo and GitHub. Hence, to log into DynaHub, you just log into your GitHub account. DynaHub will provide the interface within Dynamo.
There are two ways of logging in GitHub through DynaHub.

The first and most intuitive one is through your credentials, email and password. The alternative is to create a personal access token. This token is a generated string that can be used in place of email and password and identifies and authenticate a user.

For example, your company could create a company-wide account (we'll talk about this later in the chapter) and share it with the Dynamo users in the company. They wouldn't need their own account to get the shared material, but just this token. Of course, there are pros and cons to this approach.

If you were to follow this way, the first step would be to create the token. To do so, follow the instructions in this article: https://help.github.com/en/articles/creating-a-personal-access-token-for-the-command-line.

Also, be aware that the token identifies a user and not a repository. With the token you'll be able to access all the repositories of that user.

Click on Login to GitHub and choose the option you prefer (credentials or token).

DynaHub will then welcome you with a notification of success.

Otherwise, if you provided the wrong information, DynaHub will prompt an error message and you'll have to try again.

At this point you are in, and your Dynamo session is connected to GitHub.
Browse your GitHub content and open graphs

The great thing about DynaHub is that you can access all your GitHub content without leaving Dynamo.

If you click Browse GitHub, a pop-up window will come up prompting you to choose which repository to explore.
By expanding the dropdown selection menu, you will see a list of all the repos associated to your account, including the repos in the organisations you are part of.

Make sure you have logged in before trying to browse your GitHub content, or you will get an error. Of course, you are not allowed to see any content from an application without logging into it first. That would be weird.

Select one of the repos associated to your account.
You can select any repository from this dropdown menu and DynaHub will go through the repository and show you all the Dynamo content in it.

DynaHub shows the .dyn files in the selected repo.

At this point, you can simply **double click on the name** of the graph that you intend to use to open it in this Dynamo session.

Double click on the graph to open.
DynaHub is also able to recognise and display files in more complex folder structures. This means you can neatly organise your graphs in folders and sub-folders. This structure will be kept and displayed in DynaHub.

If there was no .dyn file in the selected repo, there is no point in showing you the content of the repo and risking you trying to open a file that is incompatible to Dynamo and crashing it. Therefore, a message will be shown instead of the content.

If you are the manager of the computational strategies of your company or organisation, it is probably useful now having a look at what happens under DynaHub’s hood when you open a graph:

Once you double click on a graph to open it, the first thing that will happen without you noticing is that DynaHub will create a temporary folder called temp in the package’s folder. As soon as the temporary folder is created, DynaHub will reach the URL where the graph lives on GitHub and it will download it in that folder.
The file just opened is “secretly” saved in the package’s folder.

Having saved the file in a folder, Dynamo will now be able to open it and let you use it. The user will be now able to change the parameters in the graph and run it as a normal Dynamo graph. Once the user is done with it, he/she will simply close down Dynamo.

While Dynamo is closing, the temp folder will be deleted from DynaHub’s folder with all its content.

This workflow happens every time you use a graph. This means that the graph will be downloaded anew every time you use it and deleted every time you finish the Dynamo session. For this reason, the user can do whatever he/she wants with the graph: change values, replace nodes, delete portions of it, delete it all and even save it at the end. None of the changes made by the user on the graph opened through DynaHub will affect the central file hosted on GitHub.

The reason why DynaHub is structured this way is because this had been identified as the best workflow to deploy graphs to groups of any dimension, without compromising the original files. Nevertheless, keep in mind that even if you did overwrite the original file in the online repo, this is GitHub and it was made for being able to track versions in time, hence you’ll always be able to roll back to a previous version of the file.

It might happen that the manager of the computational content in the organisation will make changes to a graph opened through DynaHub and he/she doesn’t want to delete the work done. Even if this is not the best approach, to save file from being automatically deleted, before closing the Dynamo session you can go to the temp folder (shown in the image above) and copy the file to any other folder, or Save As… the file to another location. In this way the file will not be deleted, you can keep editing it and even replace the original one in the repo.
Manage the packages

If you’ve used Dynamo before, you know that 99% of the times you use a graph developed by someone else, it won’t work. Probably the most common reason is that the packages employed in the graph are missing in the user’s version of Dynamo. This of course would have been a big blocker for DynaHub, which, potentially, can deploy graphs to a large number of people, not only within a company that might have standards (yeah sure, lol), but also within groups of people that don’t even know each other, such as the Australian Dynamo User Group.

For this reason, I thought it was high priority to find out a consistent strategy to deploy packages to the user.

Go back to your local repository and create a folder called `packages` and be careful when spell it, as it is case sensitive. Look in the lab’s folder to find two packages (folders with the name of two famous packages) I’ve put there for you.

The first thing to do is to zip those 2 folders, keeping their names. Copy both in the `packages` folder you just created in the local repo. Finally, open the GitHub Desktop application to commit and push the changes to the online repo.

You should now see your online repo reflecting the changes.
Now DynaHub will be able to grab the packages you stored and automatically install them. What it really does under the hood, is simply downloading the .zip files to Dynamo’s packages folder and unzipping them. Dynamo’s packages are just folders containing specific files that are placed in a special folder, after all.

Log into your GitHub account through DynaHub and then click on Get packages to pop up a window. From the dropdown bar select the repository where you have the packages you want to download. This will show a list of the packages that have been found.

Click on Get All Packages to download them from GitHub and install them. In a greenfield situation, where you don’t have those packages installed on your machine, everything will be smooth, and you’ll just receive a success notification at the end.

DynaHub will now automatically close down Dynamo for you, as you must restart it in order to correctly install the packages. Open it back once it’s closes.
If you now open the Package Manager, you will see that those two packages have been successfully installed and are now in the list.

![Installed Packages]

The packages have been successfully installed.

In case you already have the packages, DynaHub will uninstall them and re-install them. Reason being that, likely, they’re not going to be in the same version, hence they could have different content.

Follow the same process as above and this time a message will pop up notifying you that are you already have some or all of the packages.

![Notification: you already have some of the packages]

The only thing you can do now is click Ok. This will uninstall the packages that have been found and are overlapping.

Re-open Dynamo and, if you want, check from the Package Manager that the packages are not there anymore.

From here, the process is going to be the same we’ve seen at the beginning, where you log back in, click on Get packages and re-open Dynamo one last time.

This is also probably the only difference between Dynamo Sandbox and Dynamo for Revit. To fully uninstall the packages, DynaHub closing Dynamo won’t be enough. You will actually
have to close Revit and re-start both Revit and Dynamo. A bit painful, I agree, but this is the
standard installation procedure for Dynamo and there isn’t much to do about it.

At this point, it is of course the responsibility of the manager to store all the right packages in
the right versions, so the users in the group can easily use the automations.

A DynaHub strategy
To conclude this chapter, I want to briefly suggest a possible strategy for an organisation to
use DynaHub successfully. I’m not suggesting this is the best way to use it.
I’m actually keen on hearing your ideas on possible strategies or how you already use it.

The organisation could create an Organization Profile following the instructions here

The “uncomfortable Dynamo users”, those that don’t feel too comfortable in the use of
Dynamo and don’t want to get too involved with the creation of content, will be given an
access token that will allow them to authenticate as their company and browse all the
automations in the Dynamo-dedicated repos.
This should make many of these users feel a bit more comfortable, at least enough to open
DynaHub and open the automation that will save them time.

The more comfortable Dynamo users instead will have or create their own GitHub account.
The organisation will be able to add them as part of the GitHub organisation. In this way
these users will be able to see the repositories that their organisation owns, both public and
private.
The comfortable users will, in this way, be able to collaborate in the creation of a collection
automations for the whole company to use.
By branching the repository they’ll be able to structure the collaboration in a more stable
fashion and avoid painful overlaps.
How to help making DynaHub better

Do you like the project and you want to put some effort into expanding its capabilities or make the current version better? That’s amazing!

In this brief chapter we’ll talk about how DynaHub is structured and what are the next edits/additions I’m thinking of making, so that it could be easier for you to start collaborating.

DynaHub’s structure

The first thing to do, if you decide to help with expanding DynaHub, is jumping on GitHub, going to DynaHub’s repository’s URL (https://github.com/Dre-Tas/DynaHub) and fork the repository.

This has created a duplicated version of the repo on your GitHub account. Go have a look at it and click the green button Clone or download to Open in Visual Studio. The command will open Visual Studio, if you have it installed on your machine. You’ll see all the files that are on GitHub and you’ll also be able to edit them. Moreover, if you have the GitHub plugin installed in Visual Studio, you’ll be able to commit and push the changes to your version of the repo straight away, from within Visual Studio (https://visualstudio.github.com/).

If you are familiar with software architecture patterns, you’ll notice that DynaHub is structured in a MVVM (Model - View - View Model) fashion to ensure separation of concerns.

Furthermore, you can observe that there are no Models, as there is no need for data or databases, or their object representation. The Views folder contains all the xaml that defines the appearance of the interfaces for the plugin. Finally, the ViewModels contains all the logic in DynaHub, including all the methods that allow it to work.

The code in the View Model folder, ViewModels, is divided in classes that follow the existing functionalities of DynaHub:

- the login is taken care by the GitHubConnection class, containing the methods to login either with a token or with your GitHub credentials
- browsing GitHub’s content is facilitated by the BrowserEngine class that will create the repo’s tree to visualise, open a file if double-clicked and other functionalities
- GetPackages has the classes to download and install the needed packages and to uninstall the existing ones
- finally, GitHubInfo contains methods and objects that are useful for all the other classes.
The Views are organised in the same way, even though they don’t share the same names, to avoid confusions.

DynaHub’s solution structure.

Next improvements
In my personal roadmap, the first things I want to look into adding or improving are:
- Binding the data to the xaml views showing it;
- A search bar to filter the graphs by name;
- Committing and pushing files directly from Dynamo;
- Ability to create branches and pull requests directly from Dynamo;
- Visualising and approving pull requests directly from Dynamo.

Of course, this is what I think DynaHub needs to improve. If you have different priorities feel absolutely free to focus on them.

Also, keep in mind that if you think you can improve the existing codebase by cleaning or refactoring parts or the whole of it, that’s more than welcome too.