**https://youtu.be/P3aKRdUyr0s**

Npm

* Npm allows us to use pre-built code from other people instead of having to build everything from scratch.
* Stands for node package manager and is the main package manager for node js
* A package is a tool someone created and uploaded to the npm online platform
* Npm is an online platform and a command line interface (CLI).
* Graphical user interface, website

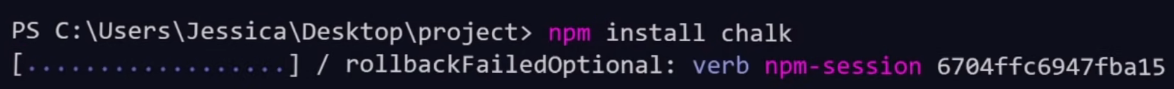
  Description automatically generated
* In the online platform, anyone can publish and share tools written in JS. These tools can be used in the browser, server, or command line.
* In the command line tool, it allows us to interact with the online platform. Via the command line tool, we can:
  + install/uninstall packages
  + use version management. Every package has a version which changes as the package changes. Npm allows us to be up to date with the version, or switch to a specific version.
  + Use dependency management. Many packages are built using other packages. Dependencies are packages that are need for another package. Npm allows us to install a package and all of its dependencies.

Installing npm

* We need to first install node js Graphical user interface, text, application, chat or text message, website

  Description automatically generated
* We can check the node and npm versions using ‘node -v’ and ‘npm -v’ respectively.
* Text

  Description automatically generated
* When we install npm packages, we can install them locally or globally.
* Graphical user interface

  Description automatically generated
* Local packages are installed only in the project folder we want to use them in. To install a package locally, we run in cmd: npm install packageName
* 
* Global packages can be used from any folder. To install a package globally, we run in cmd: npm install -g packageName
* 

Package.json

* If we are using multiple packages in our project, we can keep track of which projects we have installed via the package.json file
* A picture containing text, sign

  Description automatically generated
* This file can also store information about the project such as its name. version, author, etc
* If we want to clone a project and it has a package.json file, we can run npm install and npm will read the package.json file and install all of the packages listed in the package.json file.
* We can create a package.json file by running in cmd: npm init
* When creating this file, npm will ask us some questions about the project.
* Text

  Description automatically generated
* If we don’t want to answer the questions, we can run in cmd: ‘npm init -y’ which sets all the fields to a default value.
* When we install a package, the package.json file gets updated to include the new package we installed as a dependency.
* Graphical user interface, text

  Description automatically generated
* As well, a new folder called node\_modules is created. This node\_modules folder is where the package files are installed. We will see not only the package we installed, but other packages as well since those are the packages that our package depends on. And each of those other packages depend on other packages which are also added to the node\_modules folder
* Text

  Description automatically generated
* We can run npm list in the project folder to display all the installed packages and what versions they are on.
* Text

  Description automatically generated

Semantic Versioning

* Packages can get updated over time to add new features/fix bugs, but this can lead to conflicts between packages that the updated package depends on, or packages that depend on the updated package.
* If we type in cmd: ‘npm view packageName versions’, we get a list of all the versions of the package from the beginning
* Text

  Description automatically generated
* The versions are represented by 3 numbers, separated by a ‘.’
* The first, second, and third numbers are the major, minor, and patch versions of the package
* A picture containing text, clock, first-aid kit

  Description automatically generated
* Major updates are big and not backwards-compatible which means updating will probably break old code.
* Minor updates are backwards-compatible features and ideally won’t break old code
* Patch updates are small bug fixed that won’t break older builds
* When a new version is released, the numbers on the right reset back to 0. For example, if we are on version 1.9.15 and there is a major update, the new version becomes 2.0.0. If we are on version 1.9.15 and there is a minor update, the new version becomes 1.10.0.
* When we first locally install a package, npm will install the newest public release. When we upgrade, npm has safeguards that help prevent code from being broken.
* In the package.json file, the dependencies object will display the dependency and it’s version such as below: Graphical user interface, application, website

  Description automatically generated
* Notice the ‘^’ symbol which tells npm to update the package to the latest minor and patch version for the currently install major version. So, the above version of 4.1.1 might update to 4.2.1 for example even if version 5.0.0 exists
* We can run in cmd: ‘npm update’ to update all the packages listed to the latest minor and patch version for the currently installed major version (as shown in the bullet point above). Thus, if version 5.0.0 is released and are on version 4.1.1, we can run ‘npm’ install but nothing happens since it is a different major version.
* To override this safeguard and update to the newest version (potentially different major version), we can run in cmd: ‘npm install package\_name@latest’
* 
* To install an older/newer version of a package, we can run in cmd: ‘npm install package\_name@versionNumber’.
* 
* We can also limit updates to patches within the current minor version by running in cmd: ‘npm install [packageName@~major.minor](mailto:packageName@~major.minor)’ . So if we had version 2.2.0 installed and wanted to update to the latest patch within 2.2, we could run in cmd:
* 

Package-lock.json

* When using a package.json file from a different environment, compatibility issues can arise.
* For example, if we copy a github repo that has a package.json file and we run npm install, it is possible that a dependent package has been updated between the time the package.json file was created by the author and when we run npm install. Thus, we could be installing a different minor and patch version that was used in the original repo, which could lead to breaking of code.
* Package-lock.json solves this problem.
* If we run npm install and there is a package-lock.json file, npm will install the exact major and minor patches recorded in the package-lock.json file. This will ensure our package versions match those of the authors.