Introduction to Computer Science I
COMP 2406 – Fall 2019

Heroku and Git

Dave McKenney
david.mckenney@carleton.ca

Learning Outcomes

by the End of this Lecture, Students that have Completed the Reading Assignment and Review Questions should be Able to:

Use basic git commands to work with a project

Deploy a Node.js app to Heroku

Deploying Web Apps

Now that we can make usable web apps, the next goal is probably to allow people to use them

We will look at one way we can deploy our web apps with relative ease so they can be accessed by anybody

Deploying Web Apps

We will talk about Heroku today, but this is only one way to deploy a web app

There are MANY cloud and platform-as-a-service providers that you can deploy your apps on

You can also host your own server

Deploying Web Apps

We will discuss the barebones basics of Heroku

As the primary way of interacting with Heroku involves Git, we will also look at some basic Git commands/functionality

What is Heroku?

Heroku is a 'platform as a service' provider

Essentially, they provide a place to develop/run/manage applications

This allows you to avoid the complexities of managing servers on your own

What is Heroku?

A big benefit of platform as a service is that you can generally scale your system easily

You can provision more/less resources easily through Heroku to give your app more/less power

You pay based on the amount of resources used by your app(s)
(don't forget you have to pay in some cases…)

What is Heroku?

Apps running on Heroku are executed within 'dynos'

Dynos are "isolated, virtualized Linux containers that are designed to execute code based on a user-specified command"

What is Heroku?

This is how you can easily scale your system resources – you provision more dynos

The organization/optimization is provided by the Heroku platform

What is Heroku?

Heroku offers a free tier that you can use to host some basic apps

There are some limitations on the free tier, but if you are just getting started, it may be sufficient

What is Heroku?

You can use up to 550 'dyno hours' per month hosting up to 5 different apps

While your app is running (or apps), they take up these hours

What is Heroku?

Free tier dynos are limited to 512MB of RAM and 500MB of storeage

They also 'sleep' automatically after 30 minutes of inactivity, so the next access takes a few seconds

This helps preserve the free hours, though, so could be a benefit (especially for small personal projects)

What is Heroku?

Heroku also supports many 'add-ons'

This allows you to add additional cloud services to your app (e.g., databases, etc.)

Some of these cloud services also offer free tiers for basic setups

Heroku and Git

The primary method of deploying an app to Heroku involves using Git

For that reason, we will also look at what Git is and see some basic Git commands

What is Git?

Git is a widely used distributed version control tool

Revolves around the idea of creating snapshots of a codebase and sharing them
(with others or with yourself)

Git tracks all of the different snapshots, changes, etc. and allows you to easily switch between versions

What is Git?

When you create a local Git repository on your computer, it creates a database within a .git folder

This database will contain all the files and different versions of your code

Another common action is 'cloning' a repository from another computer

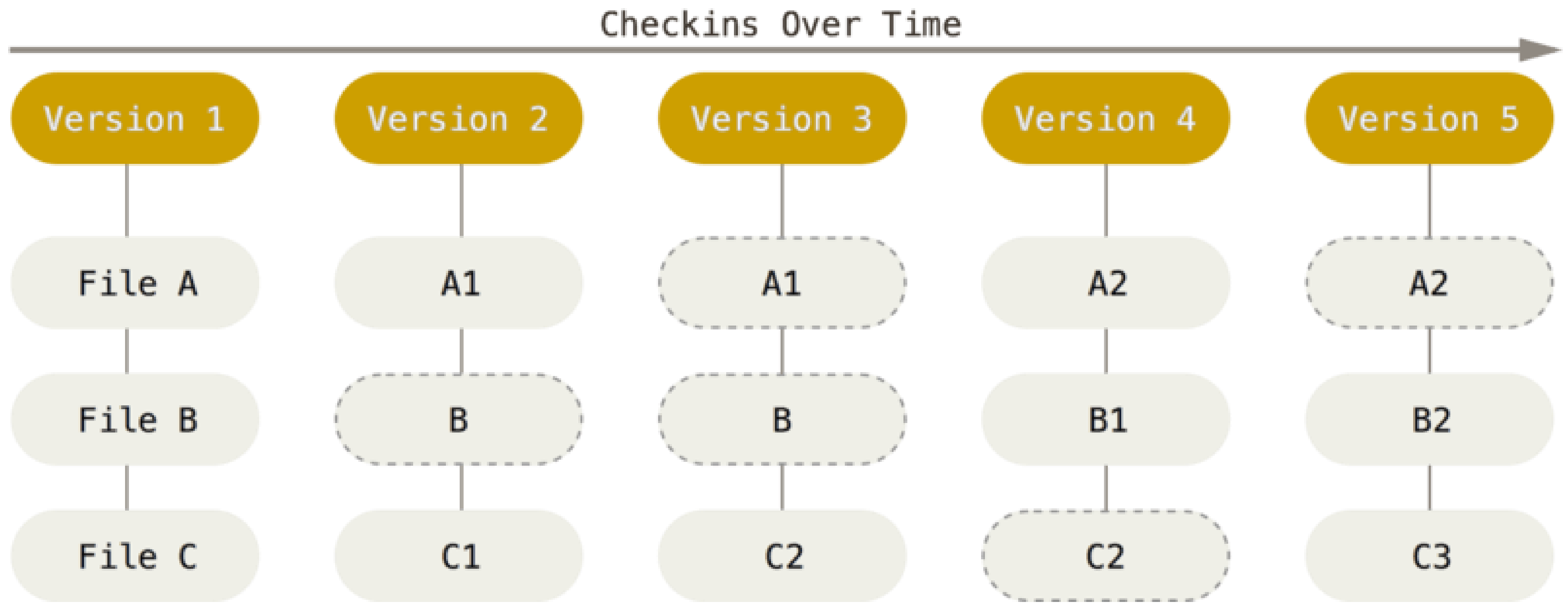
What is Git?

Any time you check-in changes to your project, Git takes a new snapshot of the project's filesystem

It stores a reference to this snapshot so you can come back to that point if you want

If files have not changed, Git does not duplicate them, it just uses a reference (this is more efficient)

What is Git?



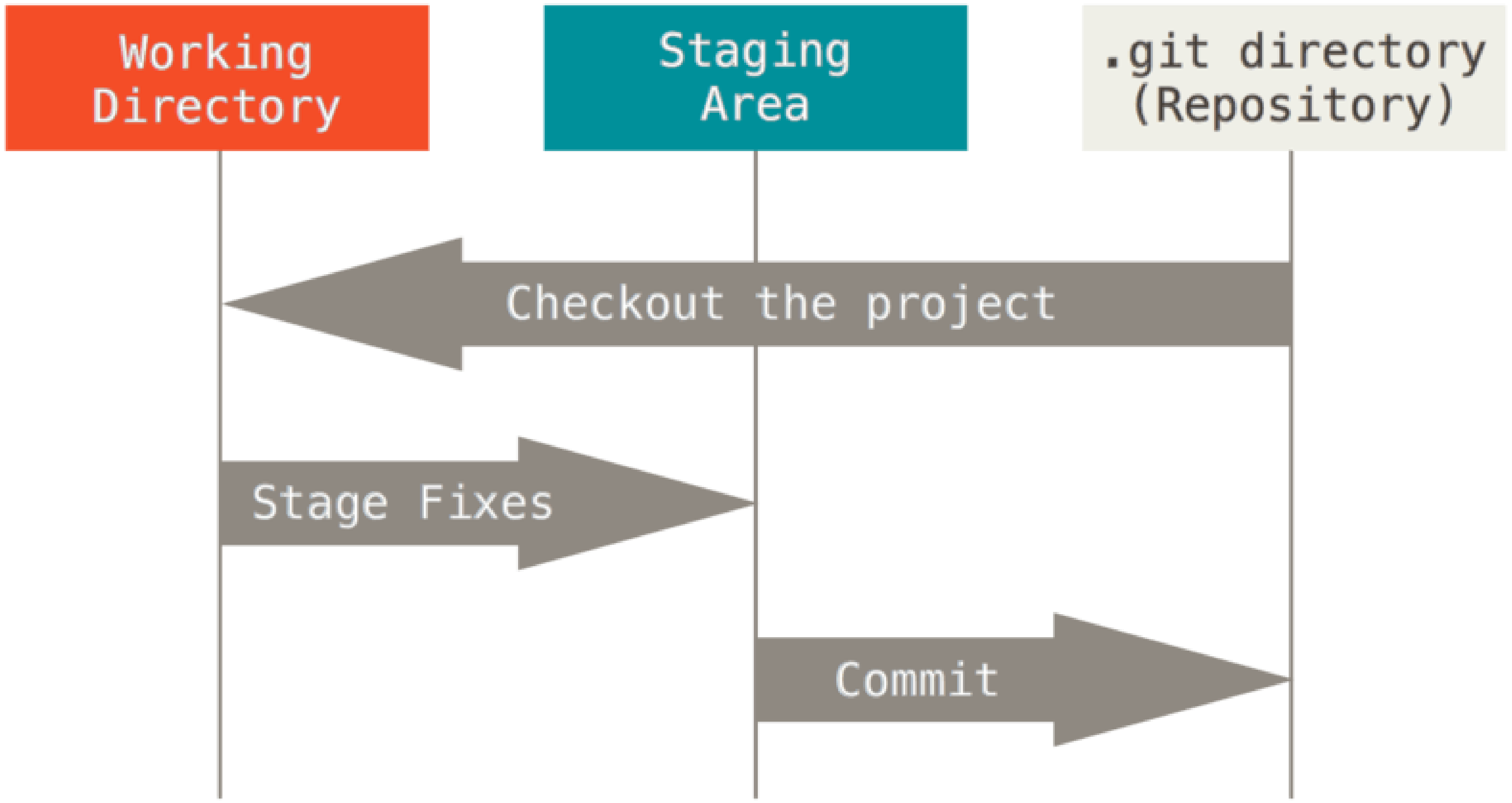
https://git-scm.com/book/en/v2/images/snapshots.png

What is Git?

Files in your Git project can be in one of three states:

1. **Modified – the file has been changed but the changes have not been committed**
2. **Staged – a modified file marked to be committed within the next commit operation**
3. **Committed – the modified file has been stored into your local database**

What is Git?



https://git-scm.com/book/en/v2/images/areas.png

What is Git?

There are three main sections to a Git project:

1. The Git directory – stores all the project data, versions, etc.
2. The working tree – this is the currently 'checked out' version you are working with
3. The staging area – stores information about what changed will be included in your next commit

What is Git?

The basic Git workflow then, looks something like:

1. Check out a particular version of the project
2. Modify files within the working tree
3. Stage the changes you want to include in your next commit
4. Perform a commit, which creates a new snapshot and stores them in the Git database

What is Git?

Again, this is just the basics of using Git

There is MUCH more to be considered

Lots of reading online if you are interested

Installing Git

Git is pre-installed on a number of systems

If it isn't on yours, you can get install information at:
https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Once Git is installed, you should be able to print out the current version using the command:

```
git --version
```

Creating/Cloning a Repository

To create a new repository, navigate to the folder of your project and issue the command:

```
git init
```

For example, we can create a git repository in the creating-git-demo folder

Creating/Cloning a Repository

You can also clone a repository from some other location

This will download all of the project information and create a local version of the repository:

```
git clone <repository_url>
```

We will use this with the Heroku example app

Working with a Repository

Within your Git repository, files can be either tracked or untracked

Tracked files are those that Git knows are part of the project – for example, files that were in the last snapshot (if there was one)

When you modify these files, Git will notice and record that they have been modified

Working with a Repository

Within your Git repository, files can be either tracked or untracked

Untracked files are any other files within the working directory

As you create new files, Git will notice these files, but will not do anything with them until you tell it to

Working with a Repository

The next command we will consider is:

git status

This shows the state of the working directory and the staging area

From here, you can see what changes have been made, which have been staged, etc.

Working with a Repository

If we run git status within the newly created repository, we will see all files are untracked

In order to have Git track the files, we have to add them into the project

This is done by adding the files (or other changes) to the staging area and then making a commit

Working with a Repository

The command to add a change to the staging area is:

```
git add <filename>
```

You can also add directories, or use wildcards:

```
git add *.js
```

Working with a Repository

Add the package.json and server.js files to the staging area

Add the public folder to the staging area

Run git status again

Working with a Repository

Along with new files, git status also shows changes that you have made to files

Edit the server.js file and run git status again

It shows the server.js file has been modified

Working with a Repository

You can git add server.js to add that change to the staging area as well

There are also options to restore the file to its original contents

Add the server.js file to the staging area again

Working with a Repository

At some point, you will have some changes that you want to save into your project

This is done by creating a new commit

Working with a Repository

The most basic way to create a commit is:

```
git commit –m "message explaining the change(s)"
```

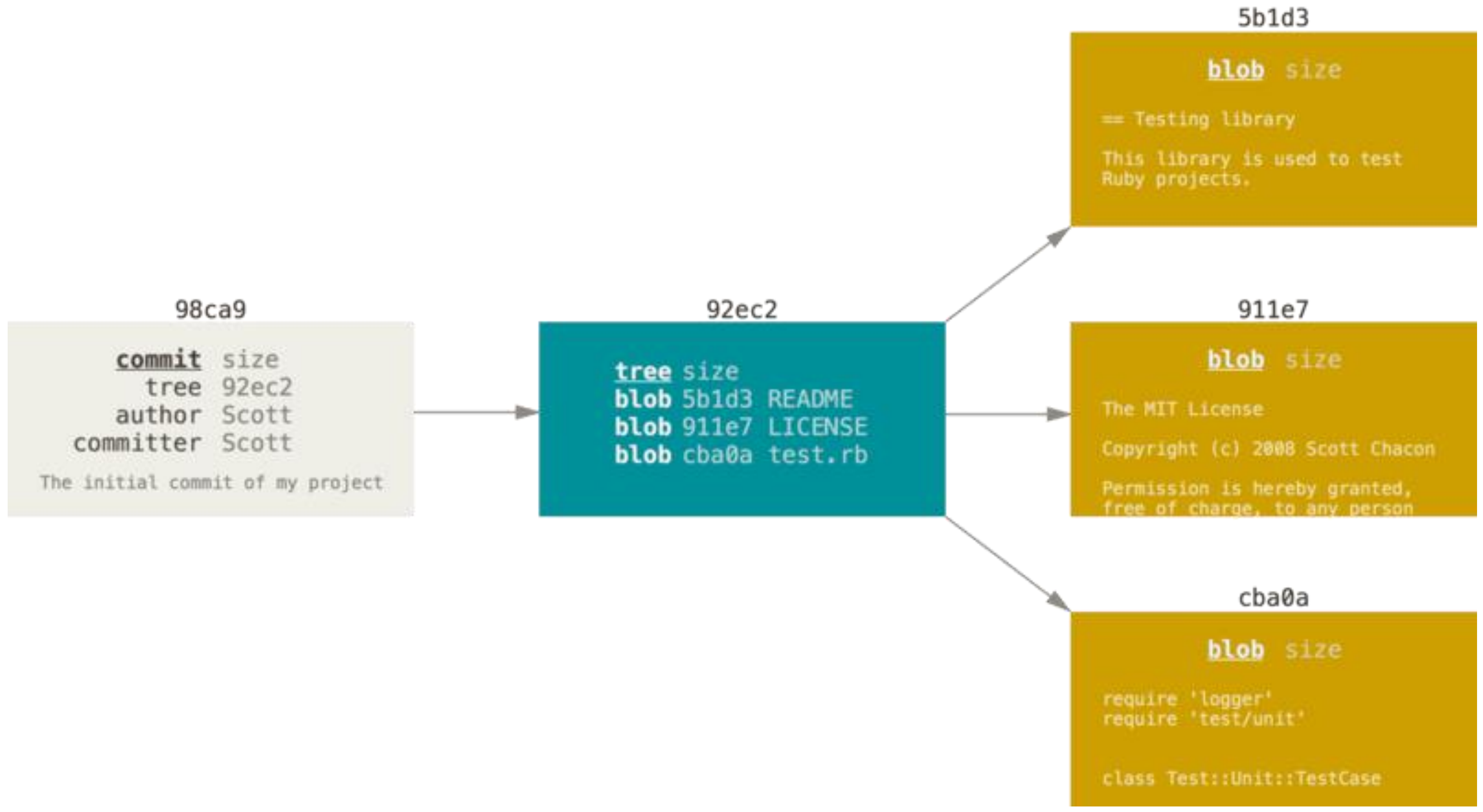
This will create a new commit containing the changes that you have marked to be staged

Working with a Repository

When you create a new snapshot with commit, Git stores the information about the files

It uses a tree structure to remember the directory structure and files within the commit

Working with a Repository



https://git-scm.com/book/en/v2/images/commit-and-tree.png

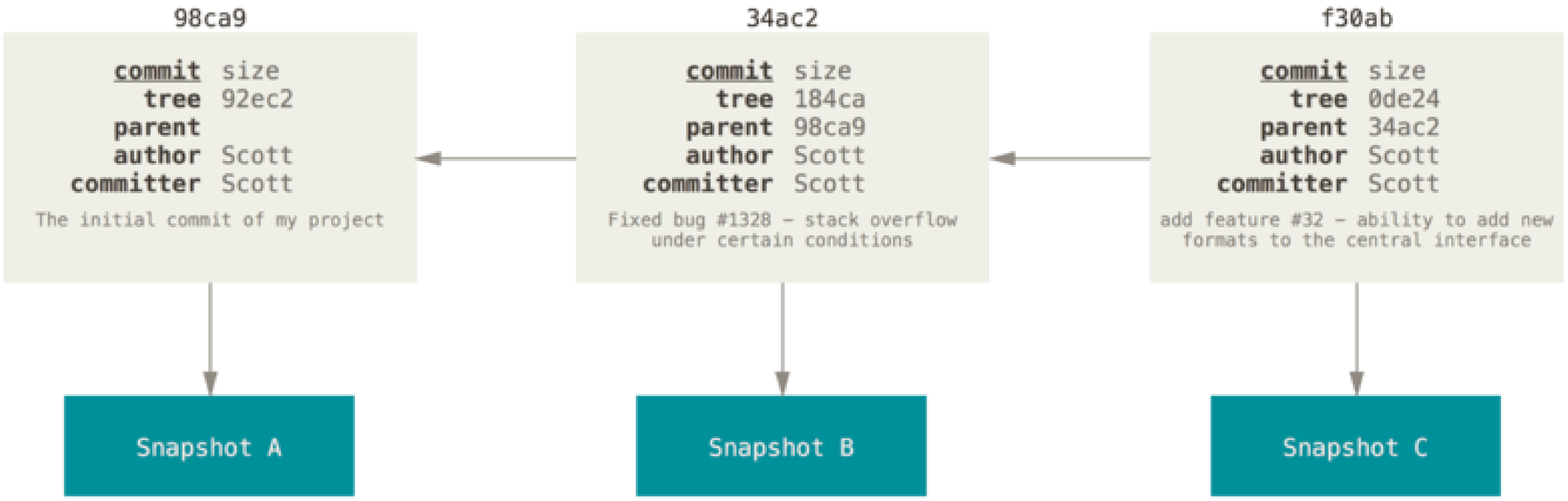
Working with a Repository

Git also remembers the order of commits

Each commit stores a reference to the commit before it (or multiple references in some special cases)

This provides a method of backtracking or tracing the change history within the project

Working with a Repository



https://git-scm.com/book/en/v2/images/commits-and-parents.png

Working with a Repository

Commit the changes in the staging area to the repository

You can use the command git log to see the history

Working with a Repository

Another common action within Git is to 'push' the changes you have made to a remote server

We will come back to this once we have a remote server to work with

Working with a Repository

At this point, we know enough Git to get our first Heroku apps working

We may also talk a bit about branching, pulling, and merging later

Knowing about these things can be useful if you are interested in working on open-source projects
(a good resume builder)

Getting Started with Heroku

To get started with Heroku, we will need:

1. **Git**
2. **Node.js and NPM**
3. **Heroku account**
4. **Heroku command line interface (CLI)**

Getting Started with Heroku

Heroku CLI install instructions:
https://devcenter.heroku.com/categories/command-line

Once installed, you should be able to get the version:

```
heroku --version
```

Getting Started with Heroku

The next step is to log in to Heroku through the CLI:

```
heroku login
```

This will take you to the browser to login

Getting Started with Heroku

As a starting point, we will use Heroku's provided Node.js starter app

We can clone the repository with:

git clone https://github.com/heroku/node-js-getting-started.git

This will download the project repository for us

Getting Started with Heroku

To create a Heroku app, run this command inside the directory your project is in:

```
heroku create
```

This creates a random name for the app and performs some other intitialization
(you can also specify an app name)

Getting Started with Heroku

The directory you ran 'heroku create' in associates that directory with the created app

So you can have multiple apps in different directories

The app the Heroku commands apply to is determined by where the commands are run from

Getting Started with Heroku

The create command output will include the URLs for the app and a remote Git repository

For example:

https://secure-reaches-30400.herokuapp.com/

https://git.heroku.com/secure-reaches-30400.git

Getting Started with Heroku

This remote Git repository is hosted on the Heroku servers

You can then 'push' your code to that remote repository to update your app at any point

Getting Started with Heroku

To push the example code to Heroku:

```
git push heroku master
```

heroku in this case refers to the name of the remote repository that you are pushing to

master refers to the branch you are pushing (we only have the master branch currently)

Getting Started with Heroku

If all goes well with the push operation, Heroku will automatically do a number of things, including:

Detect that it is a Node.js app

Setup a runtime environment for the app

Install dependencies from package.json

Start the app

Getting Started with Heroku

As this is the provided Heroku demo app, all should go well

We will discuss some additional considerations when we push our existing Git example server

Once the app has been pushed to Heroku, you can provision a dyno to actually run an instance:

```
heroku ps:scale web=1
```

Getting Started with Heroku

To access your app, you can navigate to the URL Heroku provided in the browser

You can also run this command from the CLI:

```
heroku open
```

Getting Started with Heroku

At this point we have deployed our app to Heroku

If we were fantastically rich aristocratic types, we could provision more resources for our app:

```
heroku ps:scale web=3
```

But since we are lowly free tier folk, we can't (still, this is enough for the basics)

Getting Started with Heroku

The provided Heroku example has some important steps already completed for us

If we wish to deploy our own app, there are a few steps we have to consider

We will work through deploying our original Git example app consisting of a static server

Getting Started with Heroku

#1: Heroku determines your app is a Node.js app by looking for a package.json file

So ensure your project has one and that it has all of the dependencies included

#2: Heroku automatically deploys Node.js for you on the dyno, so it needs to know what version you use

Add an entry into package.json:

```
"engines": {"node": "10.x"}
```

#3: Since the Heroku build process is automated, it needs to know how to run your server

To do so, it uses a 'Procfile' file (no file extension)

This file contains information about what services your app requires and how to start them

#3: Since the Heroku build process is automated, it needs to know how to run your server

Create a Procfile file and add this text:

```
web: node server.js
```

This tells Heroku to run 'node server.js' and that it is a web service, so it will receive HTTP traffic

#4: The dyno your server runs on will have a port dynamically assigned to it

We have been using app.listen(3000) a lot – we will have to change this for our server to work on Heroku

Why do you think the port is dynamically assigned?

#4: The dyno your server runs on will have a port dynamically assigned to it, which requires a minor change to your server code

Change: `app.listen(3000);`

To: `app.listen(process.env.PORT || 3000);`

This will use the PORT environment variable if it exists and port 3000 if it does not

Getting Started with Heroku

Commit any of the changes that have been made to Git before proceeding.

Getting Started with Heroku

To test your app locally, you can use the command:

```
heroku local web
```

This will use the Procfile to run your server and you can try accessing it at http://localhost:5000
(note: Heroku uses 5000 as the default port)

Getting Started with Heroku

Once you have tested your app locally, you can create the Heroku app within the directory:

```
heroku create
```

And then push it to the created remote repository:

```
git push heroku master
```

And provision a dyno to run it:

```
heroku ps:scale web=1
```

Getting Started with Heroku

If all goes well, your app will be deployed to Heroku

You can then access it through the browser using the provided URL or through the heroku open command

Getting Started with Heroku

The next step will be to update your server

With Git, this is easy – make changes to files, commit the changes, then push to the remote Heroku repository

Getting Started with Heroku

Once you have a basic app running, you can read through the Heroku documentation to learn about other features

There are add-ons you can attach to your app, further customizations (e.g., directing your own domain), analytics, etc.

This is Only One Way

Remember that there are many ways to host your app

With a bit of Git knowledge, along with build tools like npm, it is relatively easy to deploy to any host

Questions?

Questions?