



Introduction to Git and GitHub (for Comp 30070)

Mel Ó Cinnéide
School of Computer Science
University College Dublin

Introduction to Git and GitHub

In developing software, it's very common to want to revert to a previous version of the software.

Coordinating several developers is a related problem, though for this module you develop on your own.

Read online for further motivation for using a VCS.

These issues are usually handled by using a **version control system**.

For this module, all submitted assignment work must be developed under Git control and pushed to GitHub.

These slides will explain what you need to know.

Introduction to Git and GitHub

Git is a popular free, open source version control system, and the one you'll be using for your 30070 development.



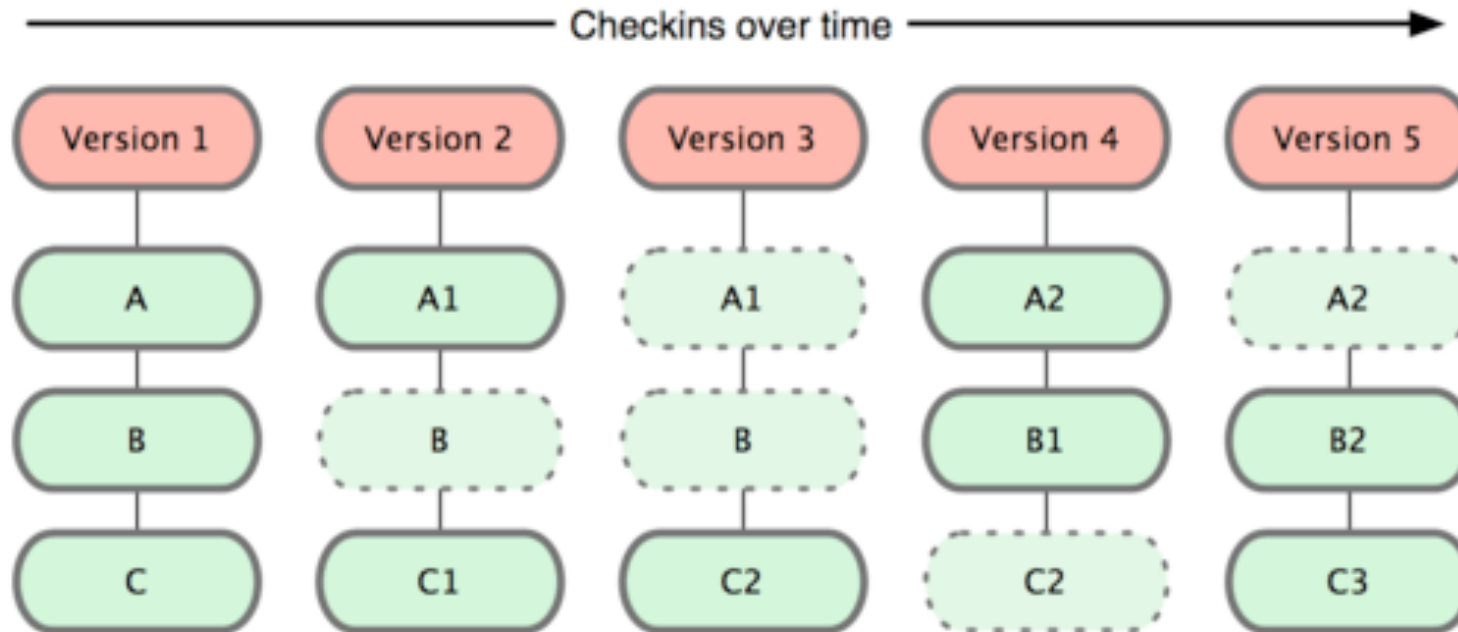
GitHub is a web-based hosting platform for Git repositories. you'll use this to share your work with 30070 staff.



In this talk we'll look first at Git, and then at GitHub.

Commits are central to Git

The key concept in Git is a **commit**. A commit is simply a version of your system, a **snapshot** of it in time.

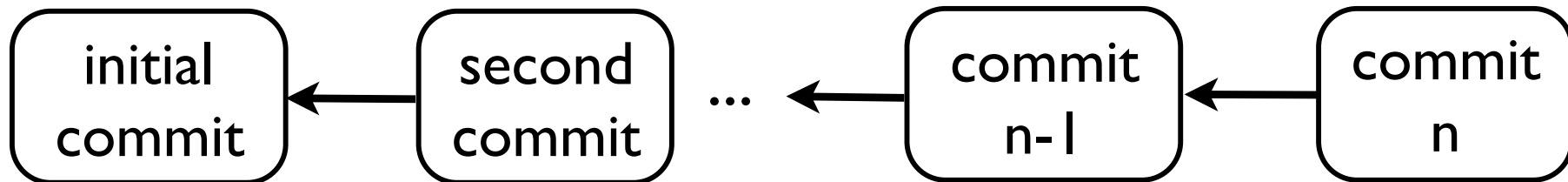


So e.g. Version 3 contains 3 files, A1, B and C2. Since A1 and B haven't changed since a previous version, a link to those files is stored rather than the file itself.

Commit History

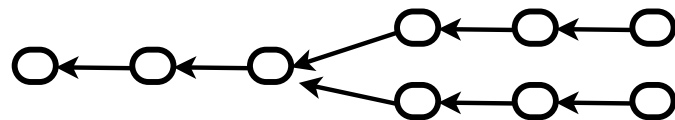
Every commit in Git, except the first, has a parent commit.

So the overall structure of a Git repository is like this:

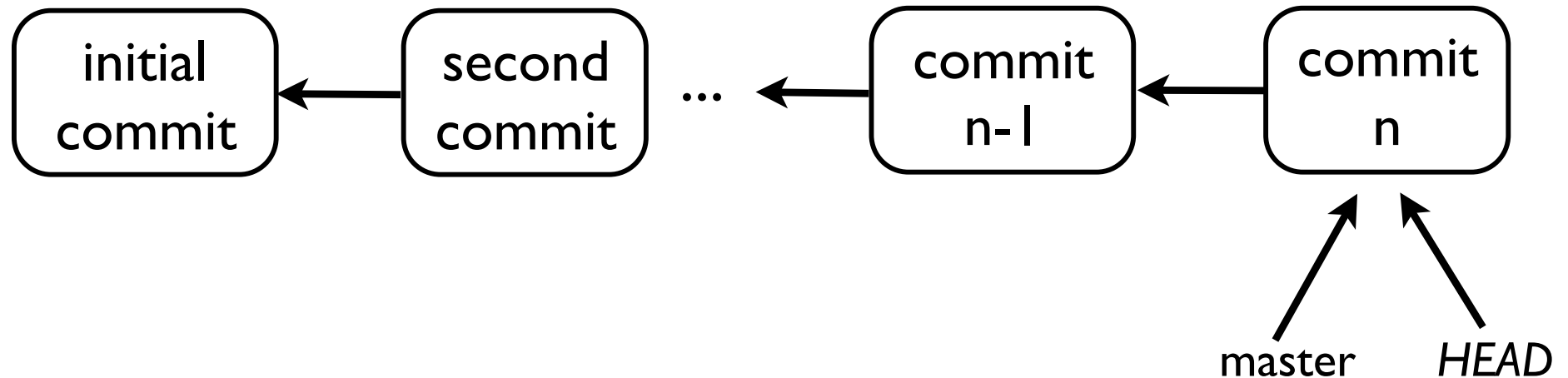


Git repositories usually have many **branches**, but we'll assume just one branch for this talk.

Source of confusion: in Git, a **branch** is essentially a **pointer to a commit**. People often use branch to mean separate development paths like this:



Some Terminology



A **head** is a reference to a commit, essentially the same thing as a **branch**.

One head is called the **master** or the **master branch**.

The currently active head is termed simply ***HEAD*** (note italics)

Performing a commit creates a new commit object with *HEAD* as its parent, and then moves *HEAD* to the new commit.

When do I make a commit?

Every time you add a bit of functionality to your program, or fix a bug, or just 'clean it up', you should commit your changes to the Git repository.

Every commit should represent a **cohesive unit of work**, and be described by a suitable commit comment, e.g.,

- * *Add display method to Clock class*
- * *Rename classes for clarity*
- * *Fix calculation bug in Terrain class*

Don't create a commit just because you're finished for the day -- a commit should make sense.

Commit often, *usually* every few hours, but it could be a few times an hour as well. (This is a much debated topic.)

Lifecycle of a file in Git

A file in Git can be one of three states:

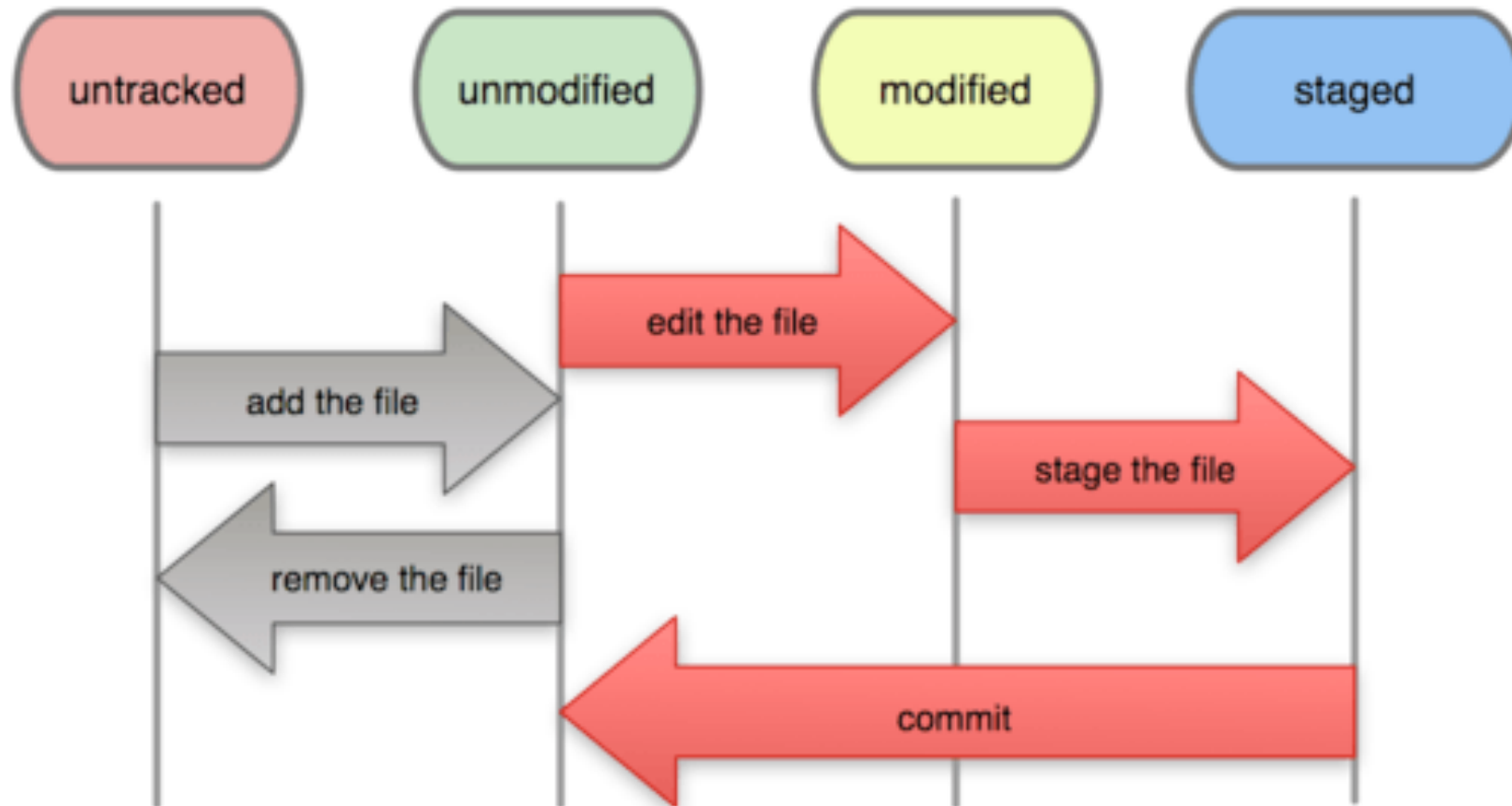
Modified: You've edited your local copy of the file and made changes, so now it's different from the version of this file you committed last time.

Staged: Staging a file means taking a snapshot of it so it's ready to be committed on the next commit. Normally, after you complete modifying a file you will want to stage it.

Committed: The file has been committed to the repository and your local copy is identical to the committed copy.

Lifecycle of a file in Git

File Status Lifecycle



An **unmodified** file is not different from its previously committed version (if one exists).

Git/GitHub for Your Project

Git is a **large** version control system. It can be used by hundreds of developers coordinating the development of a multi-million line software system.

For this module, you're working on your own developing programs of less than a hundred lines of code.

So you need only use a subset of what Git provides.

In the coming slides, I'll detail a suitable **workflow** that makes sense for this module.

Install Git

First, you need to install Git on your own computer.

See here:

<http://git-scm.com/book/en/Getting-Started-Installing-Git>

and follow the instructions in the appropriate section:
Installing on Linux/Mac/Windows (ignore the 'Installing from Source' option).

There are many graphical Git clients available. I strongly suggest to **learn Git from the command line first**.

Introduce yourself to Git

Git keeps track of who did what and when they did it, so it needs to know who you are.

Open a terminal/shell and type the following (using your own details of course):

```
git config --global user.name "Roy Keane"  
git config --global user.email "Roy.Keane@ucdconnect.ie"
```

You may use a non-UCD email address if you prefer.

You need to issue this command on each machine you'll use Git on (perhaps only one), but you need only do it **once**.

Play with Git

Before using Git for the serious work of managing your software development, you should play around with it first and deepen your understanding.

At least do the following:

- Create a new repository (`git init`)
- Add a new file to the repository (`git add`)
- Commit the file (`git commit`)
- Edit and save the file again
- View the differences (`git diff`)
- *Stage* your file (`git add`)
- Commit your file (`git commit`)
- View the commit history (`git log`)

At any stage, view the status of where you are with `git status`.

Don't just cycle through these steps mechanically. Play with them until you've a clear idea of what's going on.

Starting with GitHub

Go to GitHub (<https://github.com/>) and create your own personal account, if you haven't already. GitHub is an interesting place in its own right; have a look around.

You will be provided with a Google form that you can use to pass your GitHub ID to us.

Each assignment will be a repository in this organisation.

Every repository you create must be made private.

Your GitHub repository will be identified by an URL like:
https://github.com/UCD-Mel/3_123456789.git



Part of Mel's
organisation.

Once you have created a repository on GitHub, you won't actually *need* to visit the GitHub website again! (But it is a useful way to get an overview of your project.)

Creating a GitHub repository

PRIVATE



Owner

 UCD-Mel ▾

Repository name

/ 1_12635283 ✓

Great repository names are short and memorable. Need inspiration? How about **ducking-octo-tribble**.

Description (optional)

First 30070 assignment for Roy Keane.



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.



Initialize this repository with a README

This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

Add .gitignore: **Ruby** ▾

Add a license: **None** ▾



Create repository

Notes on Creating a GitHub repository

The name of your repository must be:

<assignment number>_<student number>

e.g., 3_12345678.

Add a suitable line of text to describe your repository. **It must include your name.**

Make the repository **private**. This is the default value anyway.

Tick the box to initialise the repository with a README (so you can clone it locally).

In the gitignore box choose Ruby. This tells Git to ignore any irrelevant files.

Creating **Git** project locally

Open a terminal/shell and navigate to the folder/directory where you want to store your project.

Now you want to **clone** the repository on GitHub:

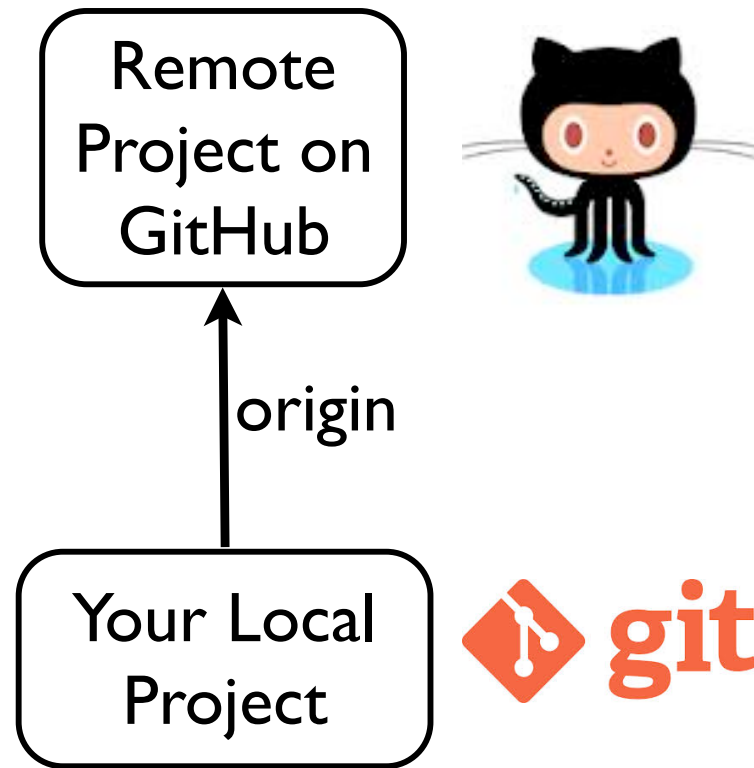
```
git clone https://github.com/UCD-Me1/1\_123456789.git
```

All going well, a folder called **1_123456789** will be created containing a file called README.md and a hidden **.git** folder.

Your local **1_123456789** Git repository is now linked to the remote **1_123456789** repository on GitHub.

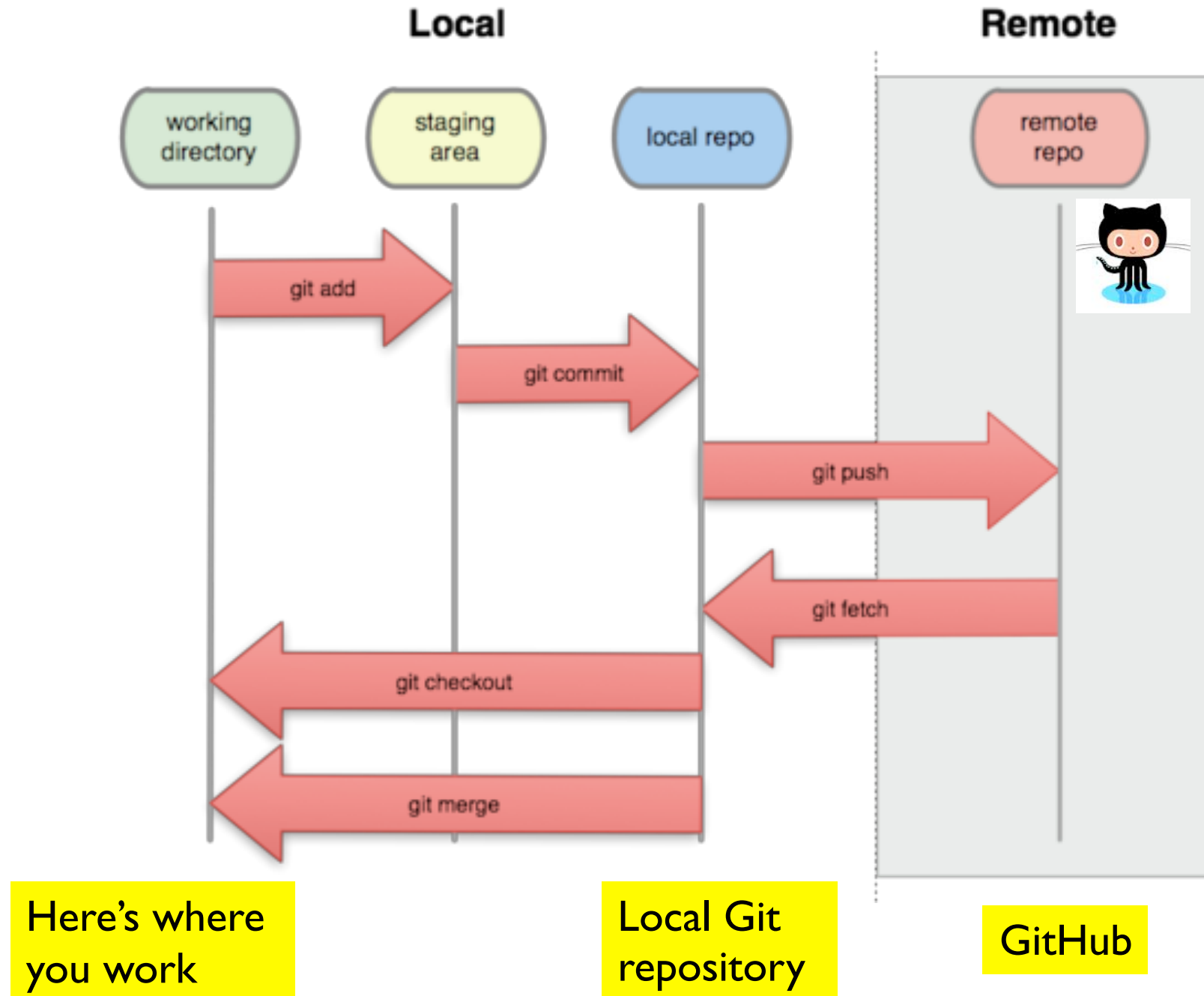
In particular, **origin** refers to this remote repository on GitHub (you'll use this in commands).

So where are we at now?



So you have your own private Git repository, and this is linked to the **remote** project on GitHub. Let's look at how that happens.

Overview of the Process



Starting Development

John gets started. Using RubyMine, he creates an empty file called **main.rb**.

To tell Git that this file is to be tracked in the repository he issues the command:

```
git add main.rb
```

This file is now *staged*.

Use “`git add .`”
to stage *all* files
you’ve edited.

To commit the staged file(s) to the repository he types:

```
git commit -m "Initial commit"
```

John now has an empty file called **main.rb** in his local Git repository. No big deal...

Extending `main.rb`

John extends `main.rb` to output some text:

```
puts 'Hello World'
```

John tests this to make sure all is working well.

To stage and commit everything to his local repository, he issues the commands:

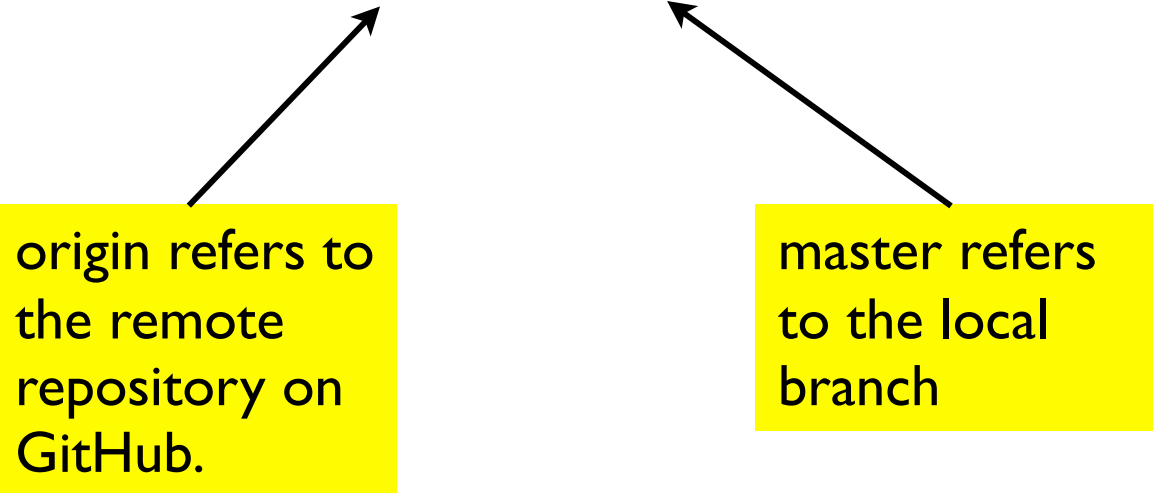
```
git add .
```

```
git commit -m "Output hello world"
```

Pushing to Github

So far, John has only interacted with this local Git repository. To **push** his changes to the remote repository on Github he types:

```
git push origin master
```



origin refers to the remote repository on GitHub.

master refers to the local branch

After this push, the COMP30070 TA and lecturer can view John's code.

Branching and Merging

As you're working on your own, you won't need to worry about creating and merging branches.

If you know how to do this and want to make use of it, that's fine.

What's happening in the repository?

The following commands are useful to see what's going in your repository:

git log: shows a log of all commits starting from HEAD back to the initial commit.

git status: shows which files have changed between the current repository state and HEAD.

git diff: shows the diff between HEAD and the current repository state.

git diff --cached: shows the diff between HEAD and the files that have been staged.

gitk: shows a nice graphical view of the repository history.

Look them up on the web or the cheat sheet.

Quick Recap

To make a local clone of a repository on GitHub use:

```
git clone https://github.com/UCD-Mel/1\_123456789.git
```

When you create a new Ruby file, put in under Git control using:

```
git add xxx.rb
```

After you finish a coherent set of updates, stage and commit all your changed files using:

```
git add .
```

```
git commit -m "commit comment"
```

To push all your changes to GitHub use:

```
git push origin master
```

There's so much more...

We have just looked at the basics of Git and GitHub. There are many more useful facilities that you'll encounter.

When you gain in confidence, explore Git further, especially the topic of **branches**. Git's branches are a big part of what makes it so popular (its “killer feature”).

If you're interested in open source software development, GitHub is a good place to start exploring.

Version control is a vital part of any software project and Git is one of the most popular version control systems, so time spent on Git will not be wasted.

Resources

<http://git-scm.com/documentation>

Everything you need to know about Git. For any detail you may be confused about, this site will give a clear answer.

<http://www.sbf5.com/~cduan/technical/git/>

is a clear description of what goes on inside Git, without getting too technical.

<http://rogerdudler.github.io/git-guide>: calls itself “Git - the simple guide.” A nice overview and very clearly presented.