Git

Version control with Git and GitHub

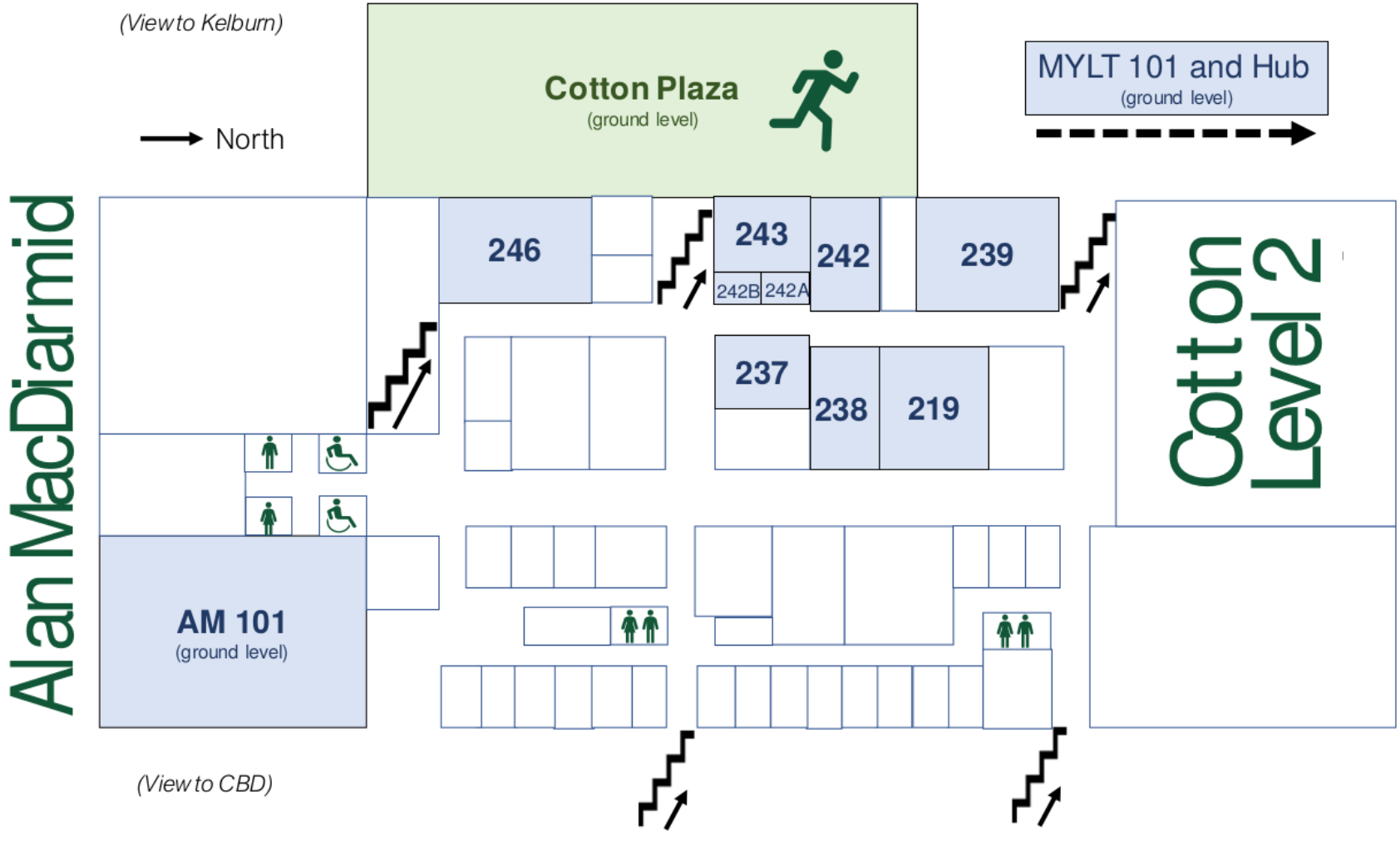
Introduction

- Evan Hanson
- evanh@catalyst.net.nz
- Please interrupt to ask questions

Administrivia

- Bathrooms
- Water
- Fire exits
- Meeting point

Administrivia



Overview

- These slides are online: http://cat-train.github.com/
- Tutorial format (with some lecturing thrown in)
- Tutorial will be in phases (time permitting)
  1. Hands-on with GitHub and Atom
  2. Questions
  3. Discussion of GitHub workflows
  4. Hands-on with Git on the command line

Overview

- Git is a powerful and potentially complex tool
- Two goals for the day
  1. Cover core concepts and terminology
  2. Familiarise with a useful workflow

Git

 https://git-scm.org/

Git

Git is a **version control system**:

software that manages different versions of files

Git

Why is it useful?

1. Keep track of changes to files
2. Review and revert back to old versions
3. Synchronise files between different locations
4. Test changes without losing the original copy

Git

Why Git and not some other software?

1. Performance
2. Flexibility
3. **Popularity**

GitHub

 https://github.com/

GitHub

GitHub is a **hosting site for Git repositories**

(with lots of extra features)

GitHub

Why is GitHub useful?

1. Free hosting for open source projects
2. Interface for browsing and editing code
3. Workflows for collaborating with others
4. API for doing other fancy things

GitHub

Why GitHub and not some other software?

1. **Popularity**

Atom

 https://atom.io/

Atom

Atom is a **text editor**

with some useful Git and GitHub-related features

Let's get started

Using Git's primary workflow

Overview

1. **Create** a project (on GitHub)
2. **Clone** the project (onto your computer)
3. **Change** some files
4. **Commit** the changes
5. **Push** the changes (to GitHub)

1. Create a project

Initialise a new "repository" from within GitHub

Create a project

- Go to https://github.com/
  - If you have an account, log in
  - If you don't, create one
    - You may need to verify your email address
- Click **New repository**
  - Or navigate to https://github.com/new

Create a project

- Enter repository name **cs4hs**
- Check **Initialize this repository with a README**
- Click **Create repository**

Remote cs4hs repository

- Exists on GitHub's servers
  - It is a "remote" repository
- Publically accessible
  - Anyone can see it
  - Only you can modify it

2. "Clone" the project

Copy the project so you can edit it

"Clone"

- Given the location of a repository
- Makes a copy of the project
- Stores it on your computer
- Links the two copies

Clone the cs4hs repository

- Open Atom
  - Close any "welcome" tabs
  - Change the theme (if desired)

Clone the cs4hs repository

- Press **Control-Shift-P** to enter a command
  - Type **GitHub: Clone** and press **Enter**
- Choose file locations
  - **Clone from:** your GitHub repository URL
  - **To directory:** wherever (but remember where!)

Local cs4hs repository

- Exists on your computer
  - It is a "local" repository
- Files and folders behave as usual
- Linked to the version on GitHub

3. Change some files

Now you can make changes to your local copy

Edit the README

- Edit file "README.md"

```
# cs4hs

## Test project

This is a project used to experiment with Git and GitHub.
```

- Save with **Control-S** (or from the **File** menu)

Add a Python program

- Enter **Control-Shift-P** and **Add File**
- Enter the path for the new file: **hello.py**

```
print("Dear world,")
print("")
print("Hello!")
print("")
print("Sincerely, Evan")
```

- Remember to save with **Control-S**

Sidenote: the ".git" folder

- The ".git" folder contains Git's data files
- They can be edited but not saved in your repository

Local changes

- You now have:
  1. Some edits to a file
  2. An entirely new file
- These changes only exist locally
- They have not been "saved" to your repository yet!

3.1 Tell Git who you are

If you saw a red error message like this:

```
*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'atomuser@WIN-VU7
```

Tell Git who you are

- Edit file ".git/config"
- Add to the end of the file:

```
[user]
  name = Your Name
  email = you@example.com
```

4. "Commit" the changes

Save your edits to a new version of the project

"Commit"

- Saving changes is a two-step process:
  1. Add them to the "staging area"
     - Tells Git to include the file in the next commit
  2. "Commit" the changes
     - Tells Git to save a new version of the project
- A "commit" is a version
  - Includes a message and attribution information

Stage the edited file

- Open the "Git" pane
- Your changes are currently "unstaged"
- **Right-click** and **Stage** "README.md"

Commit the edited file

- Enter a **Commit message**: something descriptive
- Click **Commit to master**

Stage and commit the new file

Use the same process, this time for "hello.py"

Local commit history

- You now have a series of changes
- This is your project's "commit history"
- It tells you who changed what, how, and when
- You can "revert" to previous versions

5. "Push" the changes

Now you can publish these new commits on GitHub

Log in to GitHub

- Edit file ".git/config"
- Add your username to the cloned URL

```
[remote "origin"]
  url = https://<username>@github.com/<username>/cs4hs.git
```

Push your commits

- Open the "Git" pane
- Click **Push 2**
  - Enter your password
  - Check **Remember**
- Visit GitHub to view your published changes

Review

1. Created a project (on GitHub)
2. Cloned the project (onto your computer)
3. Changed some files
4. Committed the changes
5. Pushed the changes (to GitHub)

Next Steps

Reviewing and modifying commits

Reviewing changes

Your project now has a "history" you can review

Your project's history

- A series of commits
- Every commit has:
  1. A message
  2. An author
  3. A date
  4. A set of changes
  5. A "parent" commit
  6. A unique ID
     - This is also known as its "hash" or "SHA1"

Fix-ups

What to do when you've made a mistake?

Undo a commit

- Open the "Git" pane
- Click **Undo** on the most recent commit
  - The commit is deleted
  - Its changes are moved back into the staging area
  - Its message is preserved

Fix a commit

- Edit "hello.py" and add two lines at the end

```
print("Dear world,")
print("")
print("Hello!")
print("")
print("Sincerely, Evan")
print("")
print("PS. I'm learning Git")
```

- Save the file
  - The new changes are now *unstaged*

Fix a commit

- Stage the new changes
- Commit the result
  - You have just replaced the old commit with a new one

Push the commit

- Hover over the button you used to push
  - What does it say now?

Push the commit

- To push these commits you must "force push"
- **Right-Click** and choose **Force Push**
  - This overwrites data on the remote repository!

Branches

Isolating and sharing specific sets of changes

Branches

- Each "branch" indicates a separate version
- Multiple branches in a single repository
- Let you work without interfering with others
- Eventually combined or "merged"
- The default branch is called "master"

Overview

1. Create a new **branch**
2. **Commit** to the branch
3. **Push** the branch (to GitHub)
4. Create a **pull request** (on GitHub)
5. **Accept** the pull request (on GitHub)

1. Create a new branch

Starts a new set of changes

Create a branch

- Open the "Git" pane
- Click **master**
  - Click **New Branch**
  - Type **feature**
  - Click **New Branch** (again)
- You have a new "feature" branch

2. Commit to the branch

New commits are added to the active branch

Create a new commit

- Edit the file "README.md"

```
# cs4hs

## Test project

This is a project used to experiment with Git and GitHub.

This is some information about the project.
```

- Save and commit the change as usual

Isolated changes

- Click **feature** to open the branch picker
  - Click **master** to switch branches
- README.md has changed back
  - The edit has been made to the **feature** branch
  - The **master** branch remains unchanged

3. Push the branch

Send the new branch to GitHub

Push the branch

- Open the "Git" pane
- Switch back to **feature**
- Click **Publish**
- Visit your project on GitHub
  - The **feature** branch has been pushed

4. Create a pull request

Request your branch to be "merged"

Request a merge

- Click **Compare & pull request**
- Enter some information
- Click **Create pull request**

5. Accept the pull request

Merge the "feature" branch into "master"

Merge the branch

- Make sure you're happy with the changes!
- Click **Merge pull request**
  - Add any information you'd like to include
  - Click **Confirm merge**
- Click **Delete branch**
  - Removes it from the remote repository

Review the new commits

- The two branches have been combined
- There are two new commits:
  1. Your own
  2. A "merge commit"
- The combination happened on GitHub
  - GitHub is the *remote* repository
  - Your *local* repository is now out of date

Update your local repo

- Back in Atom, open the "Git" pane
- Open the branch picker
  - Switch back to **master**
- Click **Fetch**
- Click **Pull 2**

Update your local repo

- The two new commits are pulled from GitHub
- You are back up to date with "origin"

Review

1. Created a new branch
2. Committed to the branch
3. Pushed the branch (to GitHub)
4. Created a pull request (on GitHub)
5. Accepted the pull request (on GitHub)

Merge Conflicts

When two branches can't be combined safely

Overview

1. Create two **incompatible commits**
    - One locally
    - One on GitHub
2. **Pull** the remote change (from GitHub)
3. **Resolve** the resulting conflict

1. Create two commits

Changing the same line but with different content

Edit a file locally

- Open "README.md" and change the last line

```
# cs4hs

## Test project

This is a project used to experiment with Git and GitHub.

This is some important information about the project.
```

- Make it read "important information"
- Commit the result

Edit a file remotely

- Visit your repository on GitHub
- Click the **pencil icon** on README.md

```
# cs4hs

## Test project

This is a project used to experiment with Git and GitHub.

This is some unimportant information about the project.
```

- Make it read "unimportant information"
- Enter a commit message
- Click **Commit changes**

2. Pull the remote change

Try (and fail) to combine the two branches

Fetch changes

- **Right-click** the **Push 1** button
- Click **Fetch**
  - This retrieves the new commits
- Click **Pull 1**
  - This attempts to merge the changes
- You should hit **Merge conflicts**

3. Resolve the conflict

Indicate how the two changes should be combined

Pick a side

- The two different versions are coloured
- Edit the highlighted region to be "correct"

```
# cs4hs

## Test project

This is a project used to experiment with Git and GitHub.

This is some potentially important information about the project.
```

- Remember to save the file when finished

Commit the result

- Stage the file
- Click **Commit to master**
- The conflict is resolved locally

Share the result

- We still need to push the resolution
- Click **Push 2**
- Now everything is back in sync

Review

1. Created two **incompatible commits**
2. **Pulled** to combine the two
    - Encountered a merge conflict
3. **Resolved** the conflict

Questions?

GitHub Workflows

A few helpful tips for working with GitLab

Forking

Contributing to a project that isn't yours

Forks

- A "fork" is a copy of a repo *on GitHub*
  - Copies from one user account to another
  - Think of it like a "remote clone"

Forks

- The result is a repo you control
  - You clone your forked copy
  - You push commits
- Pull request comes from your fork

Let's try it

- Navigate to my cs4hs repository
  - https://github.com/cat-train1/cs4hs
- Click **fork**

Clone your fork

- In Atom use **Control-Shift-P**
  - Choose **GitHub: Clone**
- Enter the fork's URL *including your username*
- Click **Clone**

Make some changes

- Use the standard workflow:
  1. Edit
  2. Stage
  3. Commit
  4. Push
- Visit your project on GitHub

Open a pull request

- Click **New pull request**
- Click **Create pull request**
  - This request will be in my repo

Git Culture

Some keywords to help navigate the landscape

Git Keywords

- **Squash**: Combining many commits into one
- **Rebase**: Alternative to merging
- **Feature**: Branch dedicated to specific feature
- **Hotfix**: Branch dedicated to a single bug
- **CI**: Testing new commits automatically
- Others?

Command Line

Using the Git program from the terminal

Open a Terminal

- This depends on your Operating System
  - Windows: Git Bash
  - macOS: Terminal
  - Linux: Terminal

Using the Terminal

- Quick command line how-to

The Git Command

- The **git** command (lower case) runs Git
- Enter **`git --version`** to see whether it works

Configuring Git

- `git config core.editor "atom --wait"`
- `git config user.name "Evan Hanson"`
- `git config user.email "evanh@catalyst.net.nz"`

Cloning a project

- `git clone <url>`
- `cd cs4hs`
- `git status`

Making commits

- `atom README.md`
- `git status`
- `git add README.md`
- `git commit -m "message goes here"`

Reviewing history

- `git log`
- `git show`
- `git show <commit>`

Fixing commits

- `git reset <commit>`
- `git reset --hard <commit>`
- `git add`
- `git commit`

Branching

- `git branch`
- `git branch <name>`
- `git checkout <name>`

Questions?

Thanks!

- Evan Hanson
- evanh@catalyst.net.nz
- Slides will be emailed out
- Don't hesitate to contact me!

freedom to innovate