VERSION CONTROL

with Git

Dany Vohl

Astronomy Data and Computing Services,
Swinburne

ASTRONOMY DATA AND COMPUTING SERVICES

Based on contents developed by Simon Mutch

TABLE OF CONTENT

ASTRONOMY AND COMPUTING SERVICES

INTRODUCTION

What is version control and why use it?

SCIENCE AND COMPUTING SERVICES

WHAT IS VERSION CONTROL AND WHY USE IT?

- Sometimes also called **revision control** or **source code management**
- System to record and manage file and repository changes
- Common use case:
  - *Manage **source code***
  - *Can manage changes to any kind of files*
    - Mostly those containing text
- Suitable (and viable) for both an **individual** and a **team**
  - *Enables **safe** and **efficient** collaboration on project*

ASTRONOMY COMPUTING SERVICES

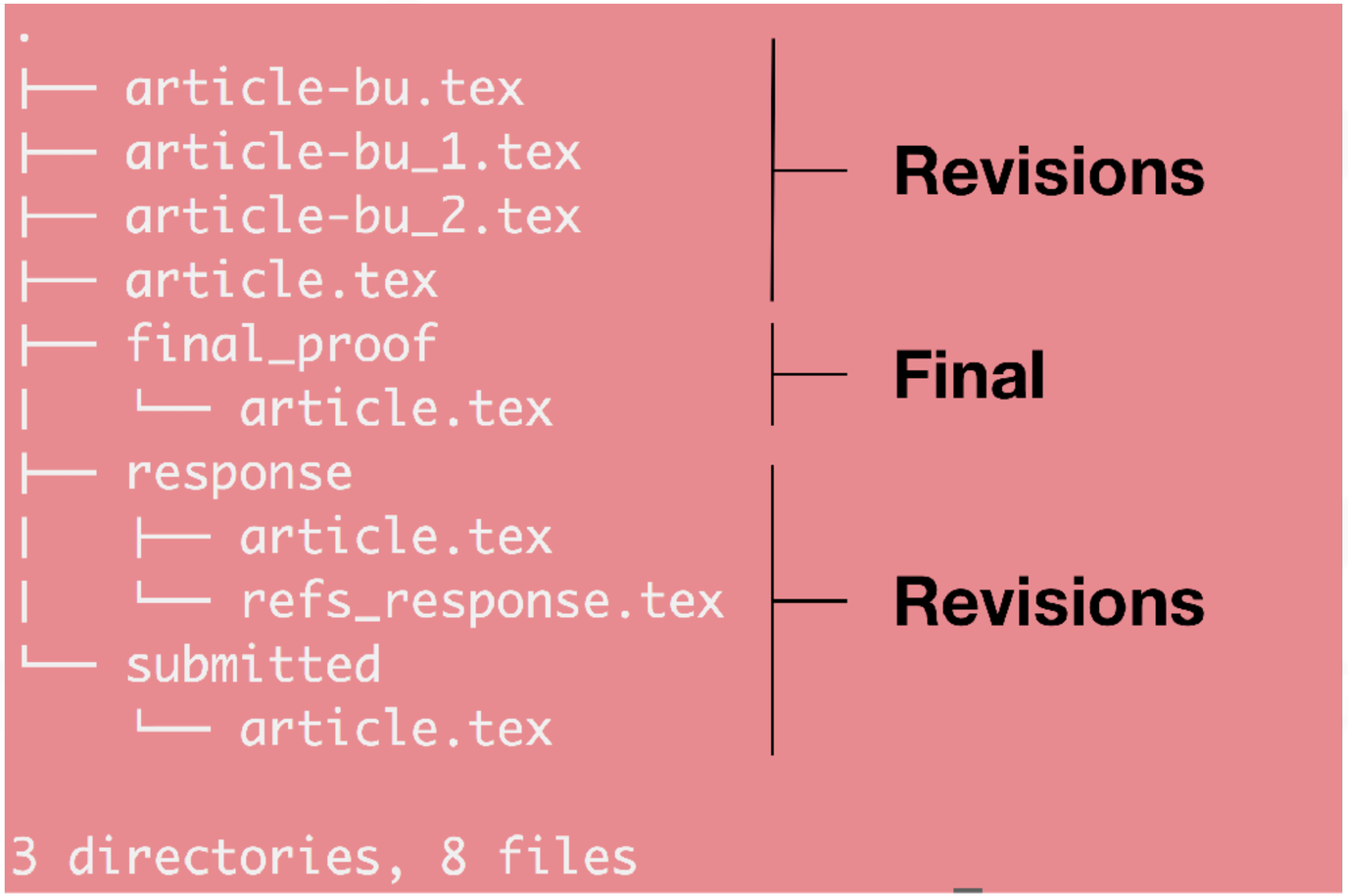
WHAT IS VERSION CONTROL AND WHY USE IT?

- You likely already use your own version control system
  - *Mechanisms to stay organised and keep track of the evolution a project*
- Many text editors like Microsoft Word, Google Docs, and Apple Pages have built-in mechanisms to track changes
- Dropbox and other online services also records changes for a time period

ASTRONOMY DATA AND COMPUTING SERVICES

WHAT IS VERSION CONTROL AND WHY USE IT?

- You probably have also employed your own simple form of version control e.g.



WHAT IS VERSION CONTROL AND WHY USE IT?

- Why using Version Control?
  - *"In practice, everything that has been created manually should be put in version control, including programs, original field observations, and the source files for papers."* *–Best Practices for Scientific Computing; Wilson et al. 2012 arXiv:1210.0530*
  - *Important aspect of any scientific endeavour is* ***reproductibility***
  - *Astronomers spend a lot of time writing code*
    - E.g. simulation or data reduction pipelines

ASTRONOMY DATA AND COMPUTING SERVICES

WHAT IS VERSION CONTROL AND WHY USE IT?

- Version control provides means to:
  - Tag code versions for later reference (via **tags**)
  - Record a unique identifier for the exact code version used to produce a particular plot or result (via **commit identifiers**)
  - Roll back our code to previous states (via **checkout**)
  - Identify when/how bugs were introduced (via **diff/blame**)
  - Keep multiple versions of the same code in sync with each other (via **branches/merging**)
  - Efficiently share and collaborate on our codes with others (via **remotes/online hosting**)

ASTRONOMY DATA AND COMPUTING SERVICES

WHAT IS VERSION CONTROL AND WHY USE IT?

- Important note
  - *Version Control's advantages are not limited to managing code*
- Also useful when writing papers, e.g.
  - *Bring back that paragraph we accidentally deleted last week.*
  - *Try a different structure and simply disregard it if we don't like it.*
  - *Concurrently work on a paper with a collaborator and then automatically merge all of our changes together.*

ASTRONOMY DATA AND COMPUTING SERVICES

WHAT IS VERSION CONTROL AND WHY USE IT?

- In brief:

> **You should use version control for almost everything.**
>
> The benefits are well worth it.

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

WHAT IS VERSION CONTROL AND WHY USE IT?

- In this tutorial, we will be using **Git** for version control

> Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

> Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.
>
> **– Git website**

ASTRONOMY DATA AND COMPUTING SERVICES

WHAT IS VERSION CONTROL AND WHY USE IT?

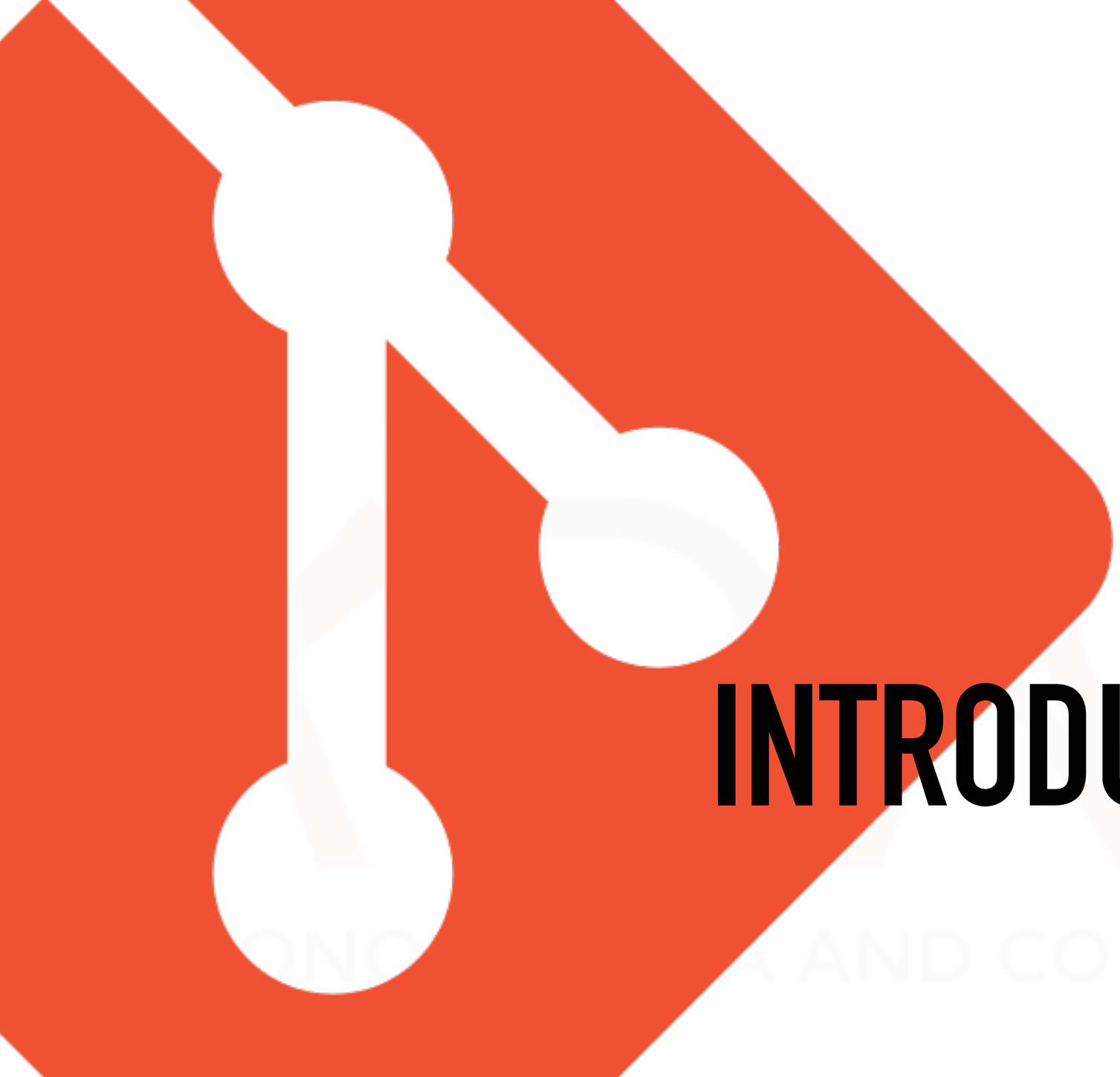
- Git's main feature
  - *It is **distributed***
  - *Every person has their own complete copy of the entire repository*
  - *Every person can make changes*
    - only commiting to, or checking out from, the "central" repository when needed
- Classic centralized systems like Subversion requires users to have access to the central repository to commit changes.
- With Git, you can easily work without a connection when you have to.

ASTRONOMY DATA AND COMPUTING SERVICES

WHAT IS VERSION CONTROL AND WHY USE IT?

- Git is also faster than many alternatives
  - *Written primarily in C and shell script*
  - *Originally written by Linus Torvalds (creator of Linux)*

ASTRONOMY DATA AND COMPUTING SERVICES



INTRODUCTION TO GIT

The basics

RESEARCH COMPUTING AND COMPUTING SERVICES

FIRST STEPS

- If not already installed on your machine, you can download an installer at http://git-scm.com/downloads
- Once installed, you have to provide your **name** and **email address**
  - *Used to sign your commits (your signature)*
  - *Provides a way to track who did what in a project*

ASTRONOMY DATA AND COMPUTING SERVICES

FIRST STEPS

- If not already installed on your machine, you can download an installer at http://git-scm.com/downloads
- Once installed, you have to provide your **name** and **email address**
  - *Used to sign your commits (your signature)*
  - *Provides a way to track who did what in a project*
- In a terminal, type the following (with obvious substitution to your info)

```
% git config --global user.name "G Lucas"
% git config --global user.email glucas@jabbaspalace.edu.au
```

ASTRONOMY DATA AND COMPUTING SERVICES

FIRST STEPS

- Next , you need to tell Git which editor you want to use by default

```
% git config --global core.editor vim
```

- Here, we set the "vim" editor as default.
  You can replace it with your favorite editor (e.g. emacs, nano, subl, …).

ASTRONOMY DATA AND COMPUTING SERVICES

FIRST STEPS

- You can enable code highlighting by telling Git to add colours to its messages

```
% git config --global color.ui true
```

ASTRONOMY DATA AND COMPUTING SERVICES

FIRST STEPS

- Now that all is set, in the next sections,
  we will proceed by working on a LaTeX document as an example project

ASTRONOMY DATA AND COMPUTING SERVICES

CREATING A REPOSITORY

- Create a repository
  - *Go to a location on your machine where you want to create the document*
    > cd path/to/my_repository
  - *Once there, create a new directory for the paper*

```
% mkdir dummy_paper
% cd dummy_paper
```

ASTRONOMY DATA AND COMPUTING SERVICES

CREATING A REPOSITORY

- Now that the repository is created, we can initialize it with git

```
% git init
```

- To verify everything has been successful, type:

```
% ls -a
```

- You should see the directory `.git`

ASTRONOMY DATA AND COMPUTING SERVICES

ADDING FILES

- Next, let's add files to track
  - *Using an editor, create and save a LaTeX file named "paper.tex" inside the newly created repository (e.g. type "vim paper.tex"), and type:*

```
\documentclass{article}

\title{A dummy paper}

\begin{document}
\maketitle

\section{Introduction}
A long time ago in a galaxy far, far away...

\end{document}
```

ASTRONOMY COMPUTING SERVICES

ADDING FILES

- Once saved, let's have a look at the status of the repository by typing

```
% git status
```

- You should see something similar to:

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#paper.tex
nothing added to commit but untracked files present (use "git add"
to track)
```

ASSIGNMY DATA AND COMPUTING SERVICES

ADDING FILES

- In order to tell Git to start tracking our new file, let's type

```
% git add paper.tex
```

- Now, typing **`git status`** again will tell us:

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#  new file:   paper.tex
#
```

COMMITTING CHANGES

- Commiting changes to the repository is the key step of version control
- This is where we save a **snapshot** of the **current state** of **all tracked files**
- A commit is done via

```
% git commit
```

- This will open the editor set by default (which we set earlier)
  - *Provide a "commit message" to describe the changes.*
  - *On the **first line**, type the following:*

```
Add basic structure of paper.tex
```

ASTRONOMY AND COMPUTING SERVICES

COMMITTING CHANGES

- Save and exit.

- That's it!
  - *We have now created a repository, added our first file, and commited our changes.*

ASTRONOMY DATA AND COMPUTING SERVICES

COMMITTING CHANGES

Tip

Writing good commit messages will make your life much easier in future when trying to track down particular changes. The first line should be a short (i.e. less than 80 characters), descriptive message that makes it clear what the relevant changes being committed are. If more detail is required then leave a blank line and add a longer more descriptive message there.

ASTRONOMY DATA AND COMPUTING SERVICES

STAGING MODIFIED FILES

- Let's add another section to **`paper.tex`**

```
\section{A New Hope}
That's no moon, that's a battle station.
```

- After saving the changes,
  running **`git status`**, you should see the following

```
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#  modified:   paper.tex
#
no changes added to commit (use "git add" and/or "git commit -a")
```

ASTRONOMY DATA AND COMPUTING SERVICES

STAGING MODIFIED FILES

- `paper.tex` falls under the category of "**Changes not staged for commit**"
- Meaning:
  - *The file has changed since the last commit*
  - *We haven't told Git that we want to include these changes in the next commit*
- To specify to include the file in the next commit, we must **stage** the file using `git add` again.
- Once running "git add", we can the current state with "git status".
  - *The file now falls under the category of "**Changes to be committed**"*

ASTRONOMY DATA AND COMPUTING SERVICES

DEALING WITH MISTAKES

- In case where, e.g.
  - *A typo has been introduced in our commit message, or*
  - *An important change was omitted before committing*
- We can easily amend our last commit using `git commit --amend`

ASTRONOMY DATA AND COMPUTING SERVICES

DEALING WITH MISTAKES

- Imagine we forgot to add the file **`bibliography.tex`** to our repository when we made our last commit.
- To fix this error, first create the file and then stage it into the index. Finally, run **`git commit --amend`**:

```
% touch bibliography.tex
% git add bibliography.tex
% git commit --amend
```

- You will then be given the opportunity to change the last commit message if you want to.

ASTRONOMY DATA AND COMPUTING SERVICES

DELETING AND MOVING FILES

- To **delete** a file in your repository, use the `git rm` command.
  - *It will both delete the file from the file system and stage this deletion action for your next commit*
- Alternatively, you can tell Git to remove a file from the `.git` repository (stop tracking the file) without actually deleting it from the file system.
  - *This is achieved by passing the `--cached` flag to the `rm` command (i.e. `git rm --cached <filename>`).*

- Similarly, to **move** or rename a file, use the `git mv` command.

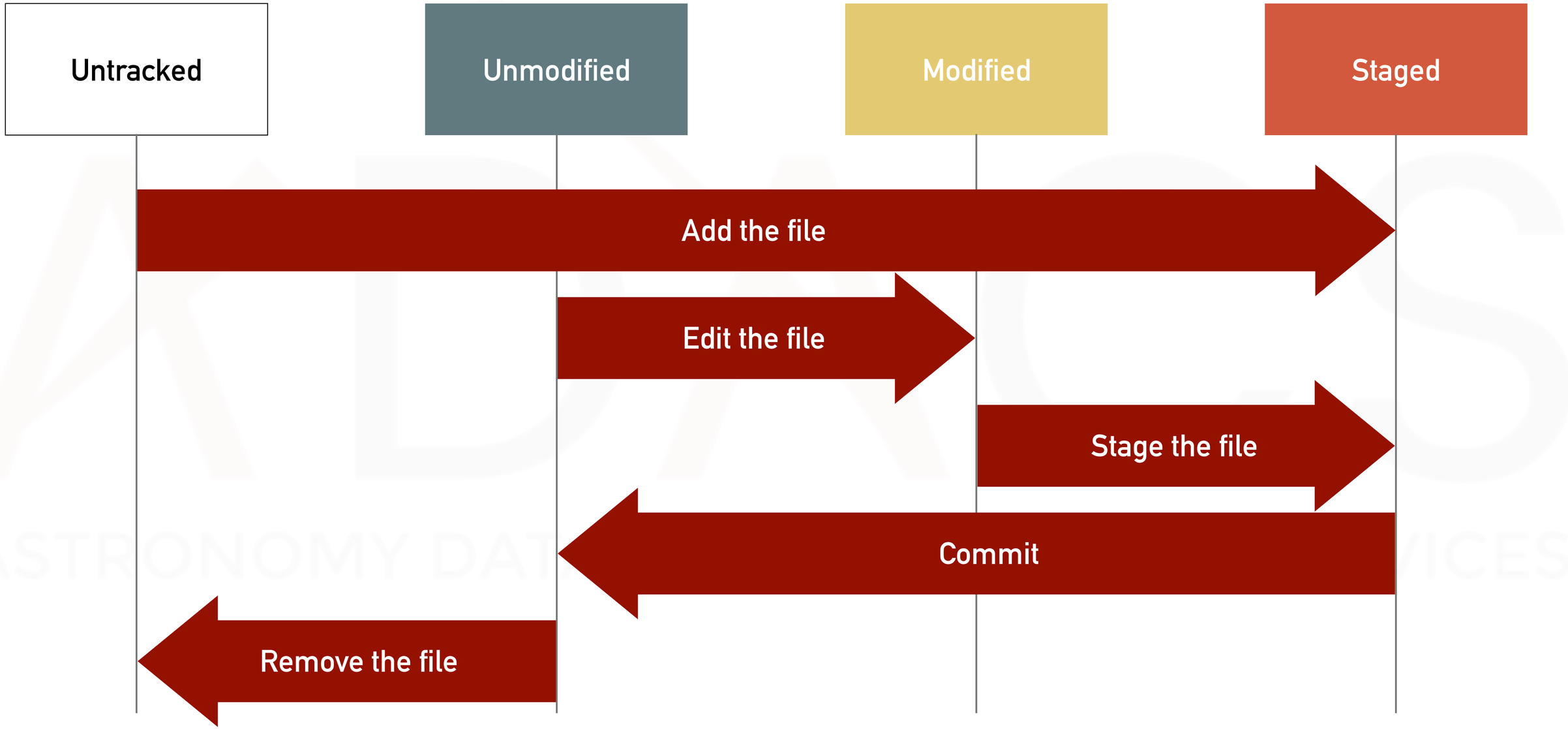
ASTRONOMY DATA AND COMPUTING SERVICES

THE CIRCLE OF LIFE

- We have covered the basic “life cycle” of files and changes in Git.
- Each file can have one of four different states:
  - ***Untracked***: *not listed in the last commit*
  - ***Unmodified***: *hasn’t changed since the last commit*
  - ***Modified***: *has changed since the last commit*
  - ***Staged***: *changes will be recorded in the next commit made*

ASTRONOMY DATA AND COMPUTING SERVICES

THE CIRCLE OF LIFE



COMMAND SUMMARY

`git init`: Initialise a new Git repository.

`git status`: Check the current status of a repository.

`git add`: Stage new and modified files.

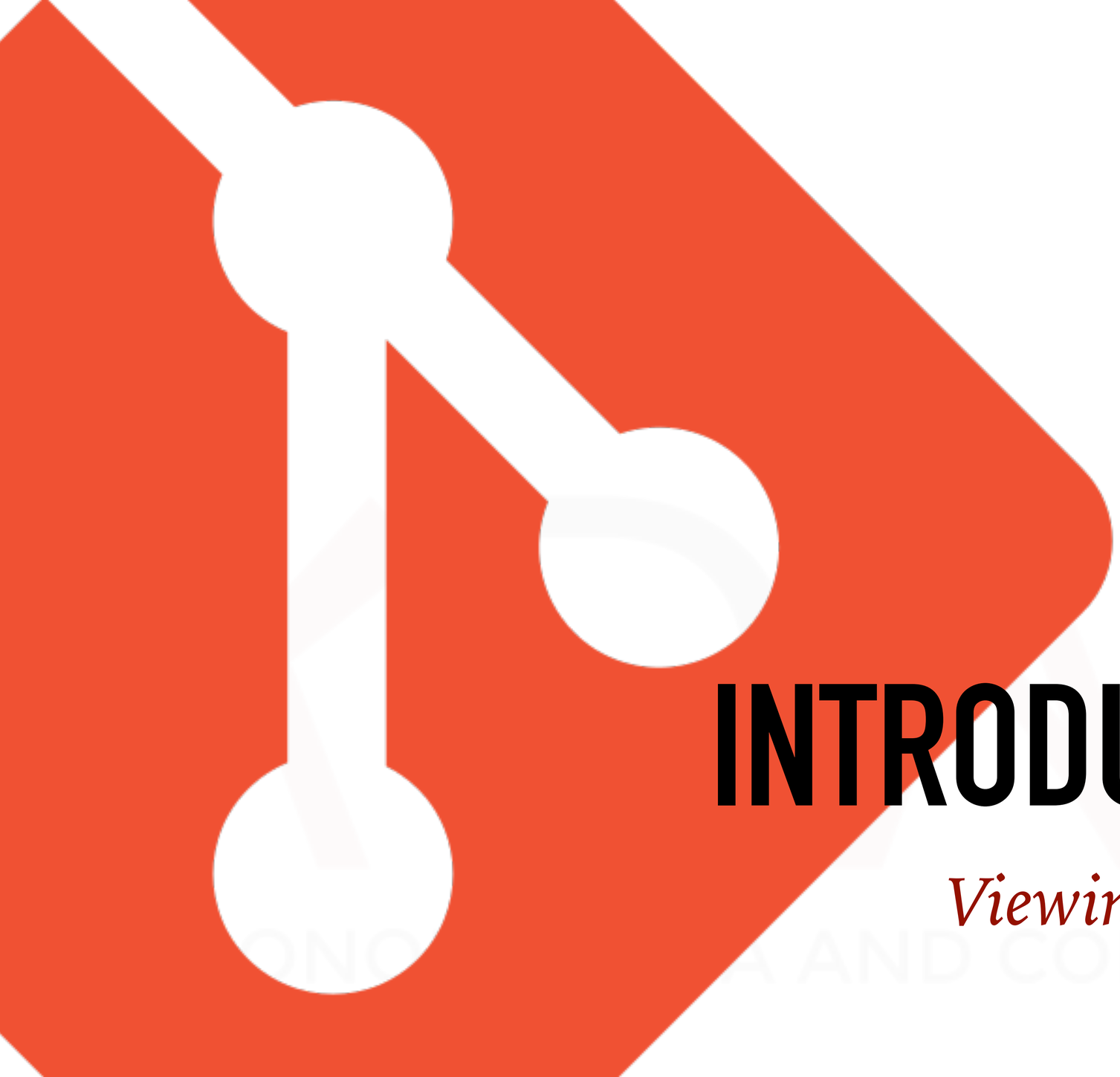
`git commit`: Commit staged changes.

`git commit --amend`: Amend the last commit.

`git rm`: Delete a file and stage this change.

`git mv`: Move a file and stage this change.

ASTRONOMY VISUALISATION SERVICES



INTRODUCTION TO GIT

Viewing and comparing commits

ONOMICS AND COMPUTING SERVICES

THE COMMIT HISTORY

- The command `git log` displays the commit history of the current branch
- Doing so in the `dummy_paper` repository, you should see something like:

```
commit 98cdaf38c12fccbfe92d4f15dc869afc12792b22
    Author: Simon Mutch <smutch@unimelb.edu.au>
    Date:   Sat Feb 16 15:56:41 2013 +1100

        Delete bibliography.tex.

    commit cc745dbfdf0421c7d84d72c75d3a52c517665fe7
    Author: Simon Mutch <smutch@unimelb.edu.au>
    Date:   Sat Feb 16 15:54:55 2013 +1100

        Add another section, appendix.tex and bibliography.tex.

    commit f615b15149a633c47f690bf891e39cb80029a71b
    Author: Simon Mutch <smutch@unimelb.edu.au>
    Date:   Sat Feb 16 15:51:06 2013 +1100

        Add basic structure of paper.tex
```

ASTRONOMY DATA AND COMPUTING SERVICES

THE COMMIT HISTORY

- As you can see, `git log` provides :
  - *the unique reference (SHA-1 checksum) for each commit*
  - *the author's name and email address*
  - *the date and commit message*
- Entries are listed in reverse chronological order (most recent first)

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

THE COMMIT HISTORY

- Many flags and arguments can be passed to `git log` to change what information is presented and how it looks. E.g.

```
% git log --pretty=format:"%h %s <%an>" --graph
```

- will result with something like

```
* 98cdaf3 Delete bibliography.tex. <Simon Mutch>
* cc745db Add another section, appendix.tex and bibliography.tex. <Simon Mutch>
* f615b15 Add basic structure of paper.tex <Simon Mutch>
```

ASTRONOMY DATA AND COMPUTING SERVICES

THE COMMIT HISTORY

- To see all different options for formatting the log output:

```
% git help log
```

ASTRONOMY DATA AND COMPUTING SERVICES

THE COMMIT HISTORY

> **Tip**
>
> ***`git help <command>`*** can be used to get the documentation for almost every Git command. If you type ***`git help`*** on it's own, you will also be presented with a list of all major commands for reference.

ASTRONOMY DATA AND COMPUTING SERVICES

THE COMMIT HISTORY

- To avoid the need to type long commands, you can define an alias:

```
% git config --global alias.lg 'log --pretty=format:"%h %s <%an>" --graph'
```

- Now, you can get the concise log view with the simple command:

```
% git lg
```

ASTRONOMY DATA AND COMPUTING SERVICES

COMPARING COMMITS

- Often, we want to compare commits to see the differences between stages
- This is done with the **`git diff`** command
- For example, to see how our paper has changed between the most recent commit and our first commit (and based on the **reference** from the log), we can type something like:

```
% git diff ef5ca0a
```

ASTRONOMY DATA AND COMPUTING SERVICES

COMPARING COMMITS

- This will provide something like the following

```
diff --git c/appendix.tex w/appendix.tex
new file mode 100644
index 0000000..e69de29
diff --git c/paper.tex w/paper.tex
index 3290236..599a0b6 100644
--- c/paper.tex
+++ w/paper.tex
@@ -8,5 +8,8 @@
 \section{Introduction}
 A long time ago in a galaxy far, far away...

+\section{A New Hope}
+That's no moon, that's a battle station.
+
 \end{document}
```

ASTRONOMY RESEARCH COMPUTING SERVICES

COMPARING COMMITS

- By specifying only one commit reference, we actually implicitely ran the command:

```
% git diff ef5ca0a..HEAD
```

  where **`HEAD`** is a shortcut for the commit reference pointing to the most recent commit.

- To access the second most recent commit, we can use the shortcut **`HEAD^`**
- These shortcuts are handy to remember when comparing commits
- **`git diff`** can also be used to see how the current state of files have changed since the last commit. To do this, simply run the command without any arguments.

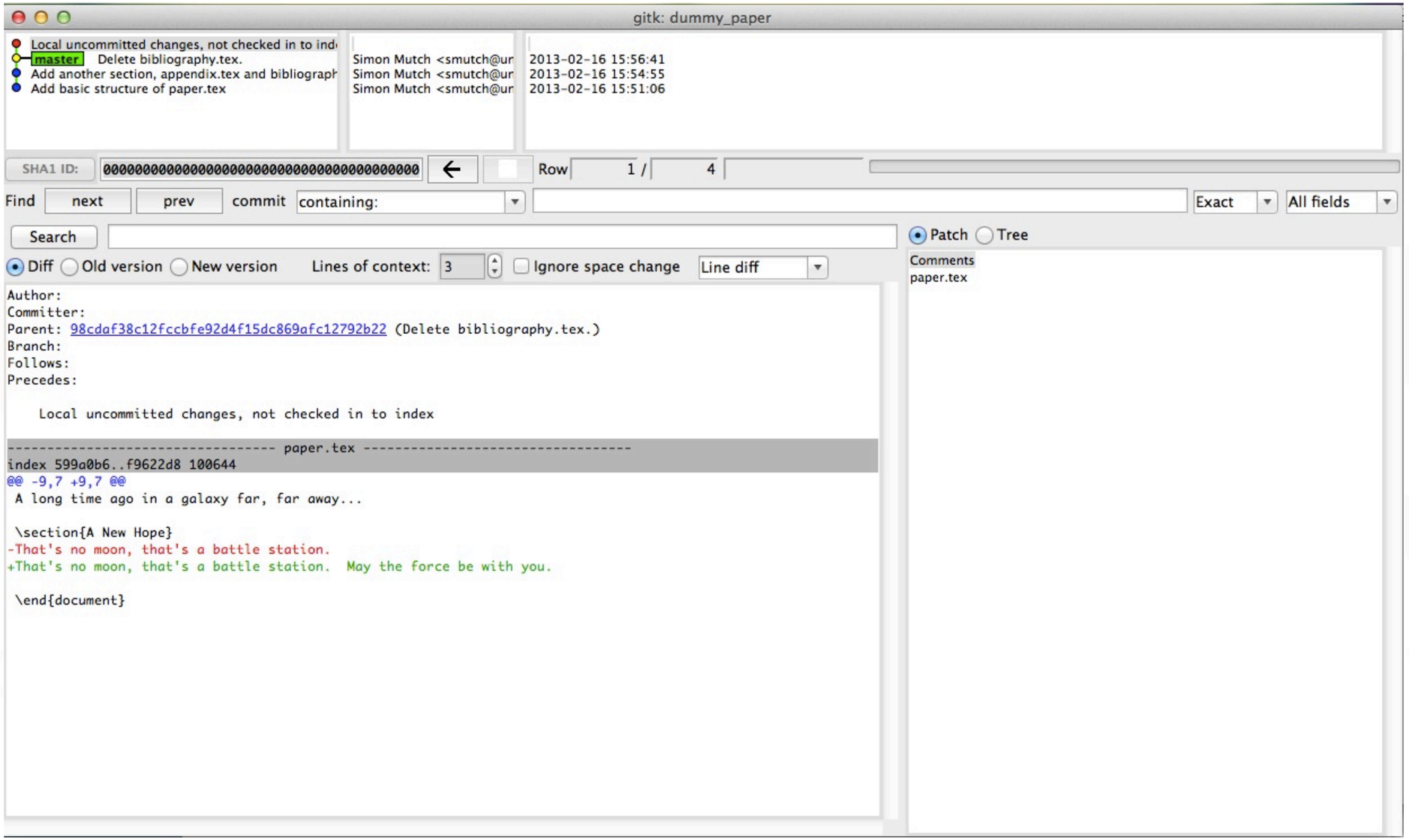
COMPARING COMMITS

- Another useful way to view the commit history is by using a Graphical User Interface (GUI) like **`gitk`** or **`GitHub GUI`**
- A list of options can be found at https://git-scm.com/download/gui/linux

ASTRONOMY DATA AND COMPUTING SERVICES

VIEWING AND COMPARING COMMITS

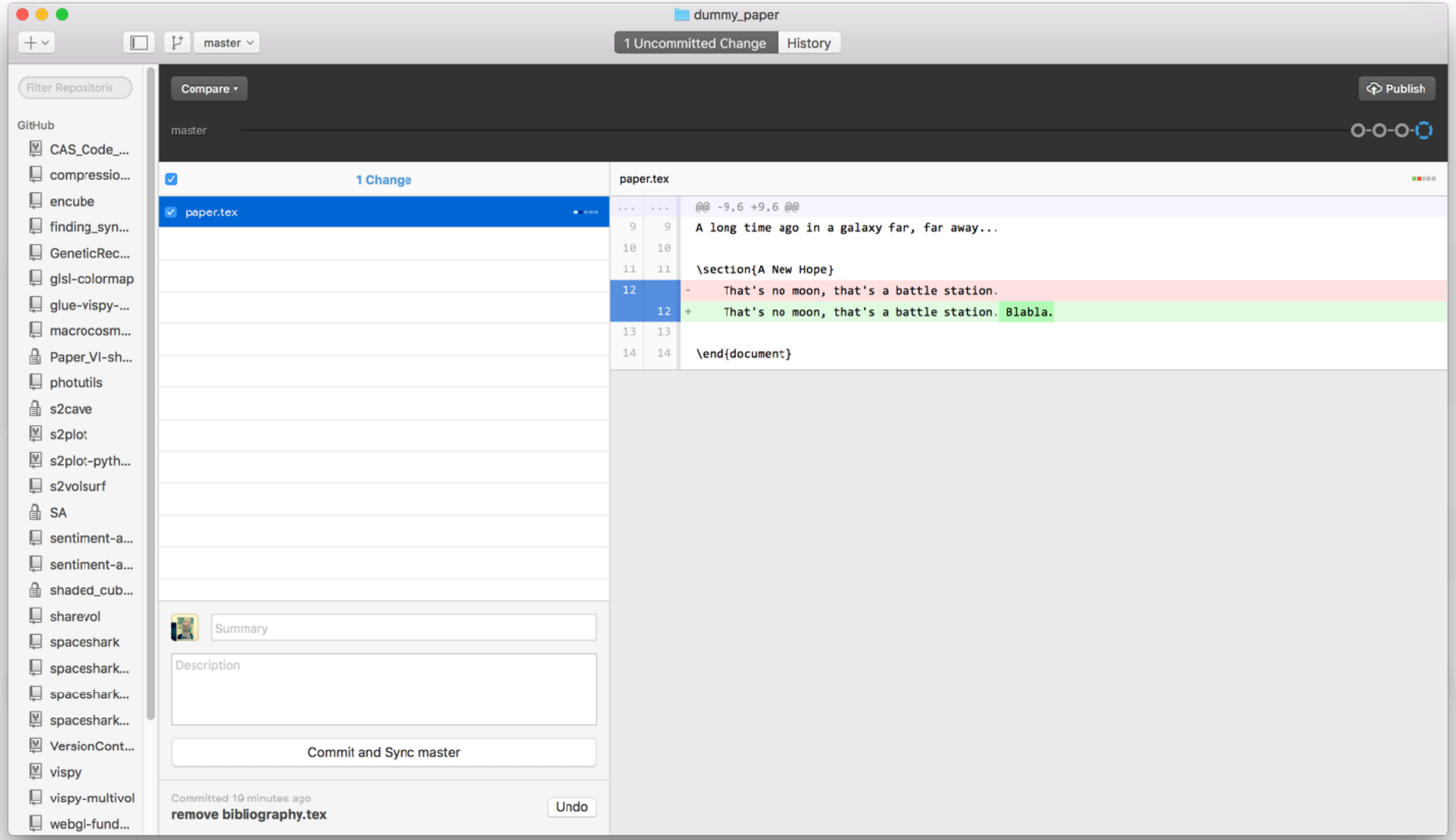
COMPARING COMMITS



The gitk GUI

VIEWING AND COMPARING COMMITS

COMPARING COMMITS



The GitHub GUI

PLAYING THE BLAME GAME

- Another useful way to visualise the history is
  - *to look at a single file and*
  - *see in which commit each line was last changed.*
- Imagine that we identified a bug in a line of code.
  - *We could then use this technique to see how long ago that bug was introduced (and by who!).*
  - *try:*

```
% git blame paper.tex --date=relative
```

and you should see a copy of **`paper.tex`** *with the reference, author and time of the last commit where each line was modified.*

ASTRONOMY DATA AND COMPUTING SERVICES

COMMAND SUMMARY

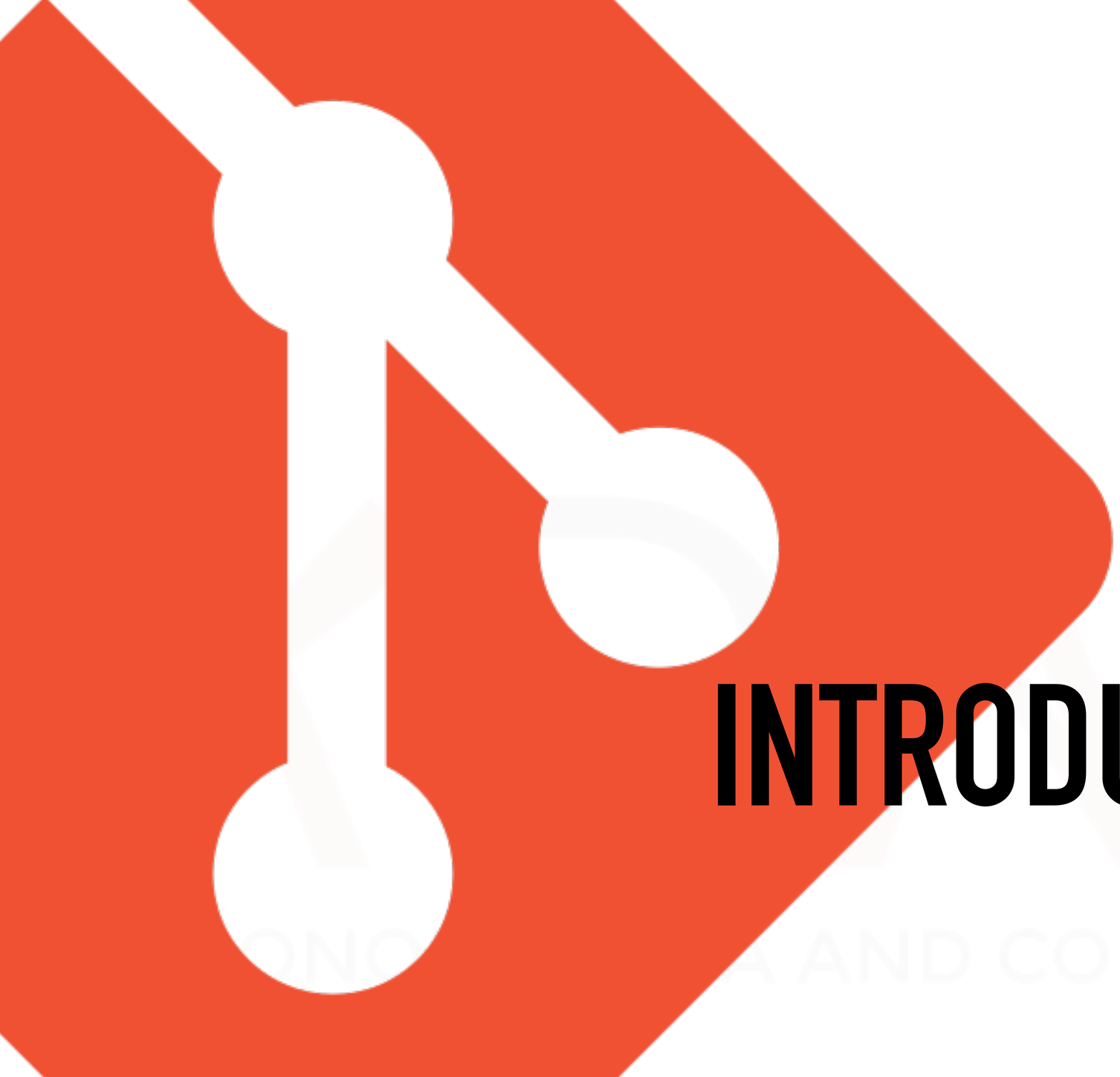
`git log`: View the commit history for the current branch.

`git diff <commit1> <commit2>`: Compare two commits.

`gitk`: View the commit history in a GUI.

`git blame <file>`: See when each line of a file was last changed.

ASTRONOMY DATA AND COMPUTING SERVICES



INTRODUCTION TO GIT

Branching

RESEARCH AND COMPUTING SERVICES

WHAT IS A BRANCH?

- Branches allow you to
  - *Diverge from your current development*
  - *Try something new without altering the history of your main work*
- For example,
  - *you could implement a new code feature whilst leaving the fully functional (hopefully working and tested) code intact for others to checkout*

- In Git, **branching** is **quick**, **flexible** and **simple**

ASTRONOMY DATA AND COMPUTING SERVICES

CREATING BRANCHES

- By default, all new repositories start on a branch called **`master`**
  - *Which can be confirmed by typing* **`git status`**

- Still in our **`dummy_paper`** repository,
  let's now create a new branch called **`risky_idea`** by typing

```
% git branch risky_idea
```

ASTRONOMY DATA AND COMPUTING SERVICES

CREATING BRANCHES

- That was simple!
- However a quick check with **`git status`** will tell us that we are still on the branch **`master`**
- To start working on the new branch, we need to perform a **`checkout`**
  - *This moves our current **`HEAD`** to the branch **`risky_idea`***

```
% git checkout risky_idea
```

- Running **`git status`** now should show that we are on the branch **`risky_idea`**

ASHER ALGORITHMS AND COMPUTING SERVICES

CREATING BRANCHES

- Running `git log`, we can see that all the previous commits from the branch `master` are still there
- When running `git branch`, the newly created branch **inherists** the history of the original branch from which it diverges.
- However, any subsequent commits to the new branch will not exist in the original (e.g. `master`)

ASTRONOMY DATA AND COMPUTING SERVICES

COMMAND SUMMARY

`git branch`: Show list of branches (and which one is active).

`git branch <branch_name>`: Create a new branch.

`git checkout <branch_name>`: Checkout (move) to a branch/commit.

`git checkout -b <branch_name>`: Create a new branch and checkout onto it.

ASTRONOMY DATA AND COMPUTING SERVICES

INTRODUCTION TO GIT

Merging and dealing with conflicts

ONOMICS AND COMPUTING SERVICES

MERGING

- At this stage we have two branches
  - ***`master`*** *and* ***`risky_idea`***
- Imagine that we have continued to work away on the **`risky_idea`** branch, committing our changes as we go
- At some stage we will want fold our changes in the **`risky_idea`** branch back into the **`master`** branch
  - *We do this by "merging" the* ***`risky_idea`*** *branch into* ***`master`***

ASTRONOMY DATA AND COMPUTING SERVICES

MERGING

- First, we checkout onto the **`master`** branch (or the one we want to merge the new changes into)

```
% git checkout master
```

- Running **`git log`**, you should see that none of your commits to the **`risky_idea`** branch are present.
- You can further confirm this by looking at the contents of **`paper.tex`**
  - *The section "The Empire Strikes Back" shouldn't be present.*

ASTRONOMY DATA AND COMPUTING SERVICES

MERGING

- Now, let's merge the branch **`risky_idea`** into our current branch:

```
% git merge risky_idea
```

- Done.

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

MERGING

- Now, let's merge the branch **`risky_idea`** into our current branch:

```
% git merge risky_idea
```

- Done.
  - *Well, most of the time.*

ASTRONOMY DATA AND COMPUTING SERVICES

DEALING WITH CONFLICTS

- Typically, merging will progress smoothly, with Git automatically working out how to merge the two branches.
- Occasionally however, this will not be the case.
  - *In particular, if we have made changes to two different branches which directly conflict with each other*
  - *then a merge will require us to tell Git which change is the correct one.*
- We will now engineer such a situation…

ASTRONOMY DATA AND COMPUTING SERVICES

DEALING WITH CONFLICTS

- First, let's create a new branch called **`episode5`** and check it out.

```
% git branch episode5
% git checkout episode5
```

ASTRONOMY DATA AND COMPUTING SERVICES

DEALING WITH CONFLICTS

- First, let's create a new branch called **`episode5`** and check it out

```
% git branch episode5
% git checkout episode5
```

- Then add another section to **`paper.tex`** with the following:

```
\section{Revenge of the Jedi}
That blast came from the Death Star! That thing is operational!
```

- And commit the changes:

```
% git add paper.tex
% git commit
```

ASTRONOMY DATA AND COMPUTING SERVICES

DEALING WITH CONFLICTS

- Now we have a new branch with a new commit that adds a section to our paper.
- However, imagine the situation where we decide we want to leave this section for the moment and go back to working on our second section.
- To do this, we return to our master branch.
- During the course of our edits we come up with another name for our newest section though, and pen this in so that we don't forget.
  - *This will lead to a conflict when we later merge our* **`episode5`** *branch back into* **`master`**. *Let's replicate this conflict now to see what happens…*

DEALING WITH CONFLICTS

- Checkout to the branch **`master`** (**`git checkout master`**)
- Then, edit **`paper.tex`** with something different than in branch **`episode5`**.

```
\section{Return of the Jedi}
    That blast came from the Death Star! That thing is operational!
```

- Again, stage and commit the changes (**`git commit -a`**)

> **Note**
>
> ***`git commit -a`*** will stage and commit the changes all at once. However, it will only stage changes in files which are already being tracked by the repository. Additionally, it will stage all changes, so you have less control over what changes go into each commit.

ASTRONOMICAL COMPUTING SERVICES

DEALING WITH CONFLICTS

- Now our two branches **`master`** and **`episode5`** have commits in them which directly conflict
- Running the **`merge`** command from the **`master`** branch will flag this conflict and Git will ask us for help (**`git merge episode5`**)
- You should see the following message telling us that a conflict occured:

```
Auto-merging paper.tex
CONFLICT (content): Merge conflict in paper.tex
Automatic merge failed; fix conflicts and then commit the result.
```

ASTRONOMY DATA AND COMPUTING SERVICES

DEALING WITH CONFLICTS

- To resolve the conflict open up **`paper.tex`** in your favorite editor.
- The offending section will look something like this:

```
<<<<<<< HEAD
\section{Return of the Jedi}
=======
\section{Revenge of the Jedi}
>>>>>>> episode5
```

- Everything between the lines **`<<<<<<< HEAD`** and **`=======`** is what exists in the **`HEAD`** commit (the tip of the master branch in this case).
- Between the lines **`=======`** and **`>>>>>>> episode5`** is what exists in our **`episode5`** branch.

ASTRONOMY DATA AND COMPUTING SERVICES

DEALING WITH CONFLICTS

- To resolve the conflict, simply pick which section is good, and delete the rest (including the `<<<<<<< HEAD` and other markups).
- After you have edited and saved `paper.tex`, finish the merge by staging and committing your results as usual (e.g. `git commit -a`)
- The commit message will be auto-populated for you, and so there is no need to edit it.
- Once merged, if you do not need the other branch, you can delete it using `git branch -d`

ASTRONOMY DATA AND COMPUTING SERVICES

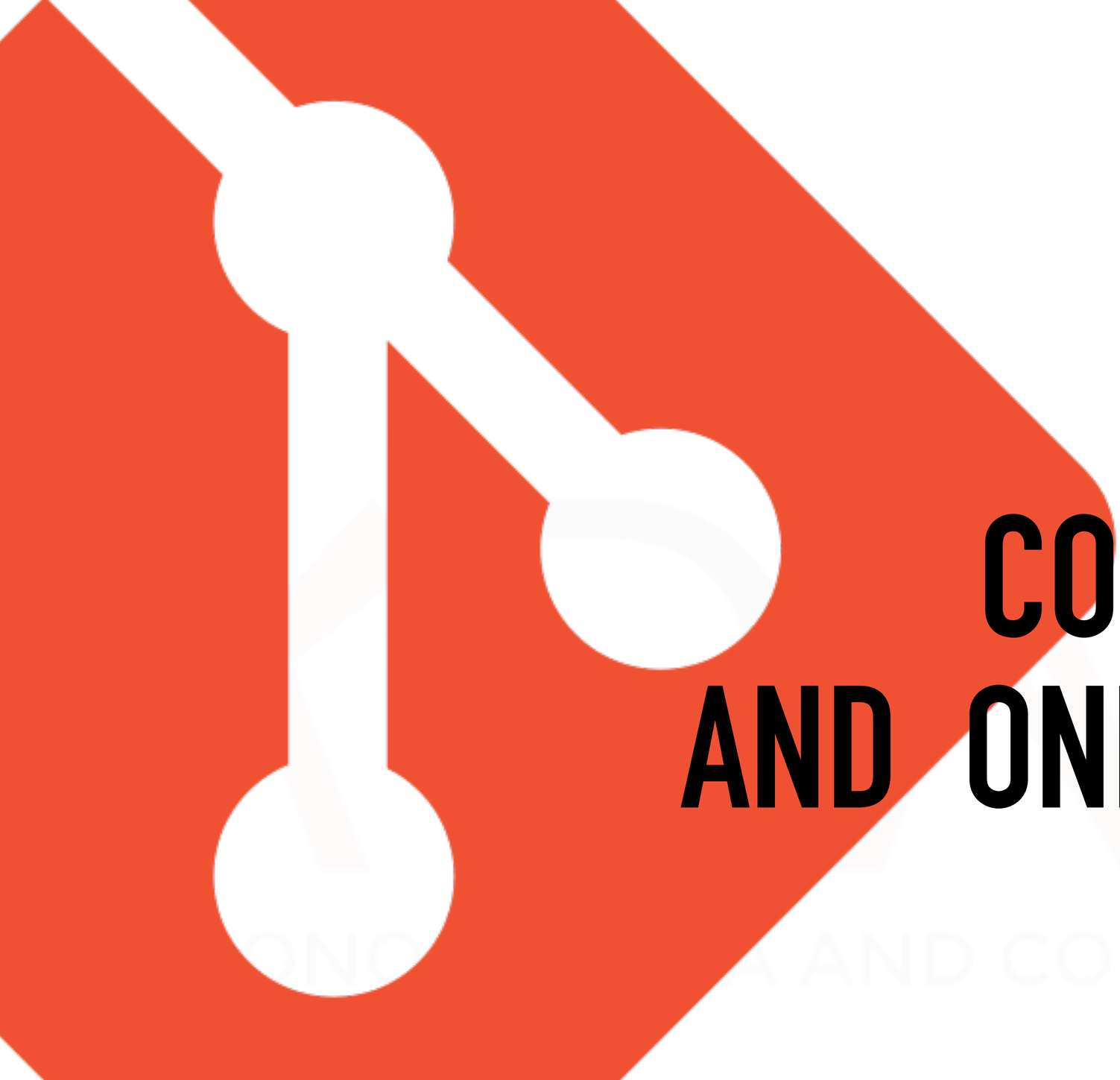
COMMAND SUMMARY

`git merge`: Merge branches and commits

`git branch -d <branch_name>`: Delete a branch branch.

`git commit -a`: Stage **all** changes in **tracked** files and commit them.

ASTRONOMY DATA AND COMPUTING SERVICES



COLLABORATION AND ONLINE HOSTING

BitBucket, GitHub, …

ONOMICS AND COMPUTING SERVICES

COLLABORATION AND ONLINE HOSTING

- As well as managing our own codes and documents, another important use of version control is its **collaboration capabilities**
- As mentioned previously, Git uses a “distributed” model
  - *Allowing everyone working on a project to have their own independent copy of the entire repository*
- To collaborate effectively, we need a **central version of the code base**
  - *Used to unify everyones’ efforts*
- Typically, the best location for a central repository is **online**

ASTRONOMY DATA AND COMPUTING SERVICES

HOSTING

- There are a number of excellent options for online hosting of git repositories.
- Currently, two options are dominating the landscapes:
  - *Bitbucket (https://bitbucket.org/)*
  - *GitHub (http://github.com/)*

ASTRONOMY DATA AND COMPUTING SERVICES

HOSTING

- There are a number of excellent options for online hosting of git repositories.
- Currently, two options are dominating the landscapes:
  - ***Bitbucket*** *(https://bitbucket.org/)*
    - This site offers unlimited free public repositories (where anyone can see and checkout your project).
    - With an email address from an academic institution will also get you unlimited free private repositories!
    - These repositories only allow users who you specify to have access.

ASTRONOMY DATA AND COMPUTING SERVICES

HOSTING

- There are a number of excellent options for online hosting of git repositories.
- Currently, two options are dominating the landscapes:
  - ***GitHub*** *(http://github.org/)*
    - This is site also offers unlimited free public repositories.
    - An academic institution email address will get you 5 free private repositories.
    - Github is probably the place for new open source software and tools.
    - It's a fantastic service and well worth using, especially if you want to take your own code open source.
    - You can also use Github to serve web pages for free.

ASTRONOMY DATA AND COMPUTING SERVICES

CLONING A REPOSITORY

- If you have the address (and correct permissions) for an online repository
  - *You can grab your own copy using the clone command*
- Try cloning your own copy of the source for this tutorial (make sure you are not in your a_paper repository when you do this):

```
% git clone git://github.com/AstroinformaticsAU/VersionControlTutorial.git
% cd VersionControlTutorial
```

  - *You are now inside your own personal copy of the repository and can do whatever you want with it. Try:* `% git log`
  - *You will see that you also have the full commit history.*

ASTRONOMY DATA AND COMPUTING SERVICES

COLLABORATION STRATEGIES

- Covering all of the different ways you can collaborate with Git goes out of scope of this tutorial
- A number of options on how to get your changes incorporated into the central repository exist
- These include:
  - *Forking and pull requests*
  - *Email patches*
  - *Direct pushing*

ASTRONOMY DATA AND COMPUTING SERVICES

COLLABORATION STRATEGIES

- The basic work-flow is almost always the same though:
  - *Make your changes in your own personal copy of the repository, ideally in a new branch.*
  - ***Pull** (using the command `git pull`) the most recent version of the central repository into your master branch*
    - This makes sure you are up-to-date with any changes made by someone else subsequent to when you last pulled (or made your original clone)
  - ***Merge** your changes from your new branch into master*
  - *Once any conflicts are resolved you can update the central repository by **pushing** your new code onto it (using for example `git push`)*

ASTRONOMY DATA AND COMPUTING SERVICES

FURTHER READING

- For a proper introduction to hosting and collaborating with Git, see the excellent online book, Pro Git (http://git-scm.com/book)
- The help pages of Github are also an excellent resource

ASTRONOMY DATA AND COMPUTING SERVICES

MORE RESOURCES

- Other great tutorials and resources for learning Git include:
  - ***GitHub online interactive tutorial***
    - https://try.github.io/
  - ***The Git Pro book***
    - http://git-scm.com/book
  - ***The Github help pages***
    - https://help.github.com/
  - ***John McDonnell's Git for Scientists tutorial***
    - http://nyuccl.org/pages/GitTutorial/

ASTRONOMY RESEARCH COMPUTING SERVICES

COLLABORATION AND ONLINE HOSTING

GIT CHEAT SHEET

- A good Git's command overview provided by GitHub is available at https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf
- For another good `cheat sheet'-style website, I encourage you to have a look at http://ndpsoftware.com/git-cheatsheet.html

ASTRONOMY DATA AND COMPUTING SERVICES