



UNSW  
SYDNEY



CENTRE FOR  
BIG DATA RESEARCH  
IN HEALTH

# HDAT9800

# Visualisation and Communication of Health Data

Chapter 1 - git and GitHub

---

Drs James Farrow & Tim Churches



# Communication is a process

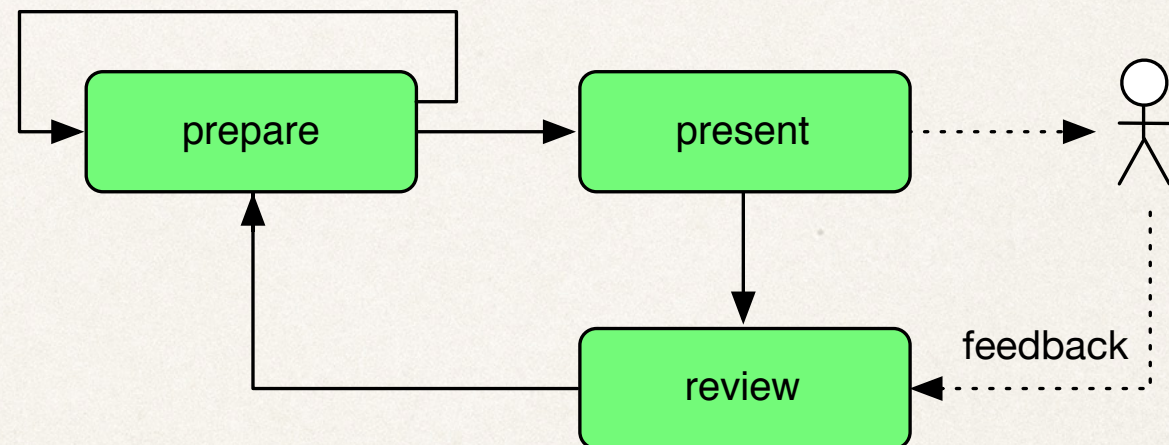
---

Communication begins before the presentation

Communication continues after the presentation

An iterative process

- ❖ prepare (iterative)
- ❖ present
- ❖ feedback
- ❖ review (iterate back to beginning)





# Revision management

---

We need to manage the artefacts of our communication

Iteration means revision

Revision means multiple versions

The content of any project will change over time



# A typical project

---

Ad hoc management of files

Commented out chunks of old code in scripts

Copies of older versions 'just in case'

Typically using suffixes on files and directories

- ❖ .new, .old, .orig, .new.old, .new.new, .new2
- ❖ oldproject, project.jun, project.0410

A real mess



# Shift in thinking

---

Instead of taking copies 'beside' one another...

...we're going to keep a history of changes

This will let us delete 'old' documents, data and code

Everything is recoverable

Our projects stay 'clean'

We 'save' every once in a while 'just in case', this is no more intrusive than that once you're used to it



# Version control systems

---

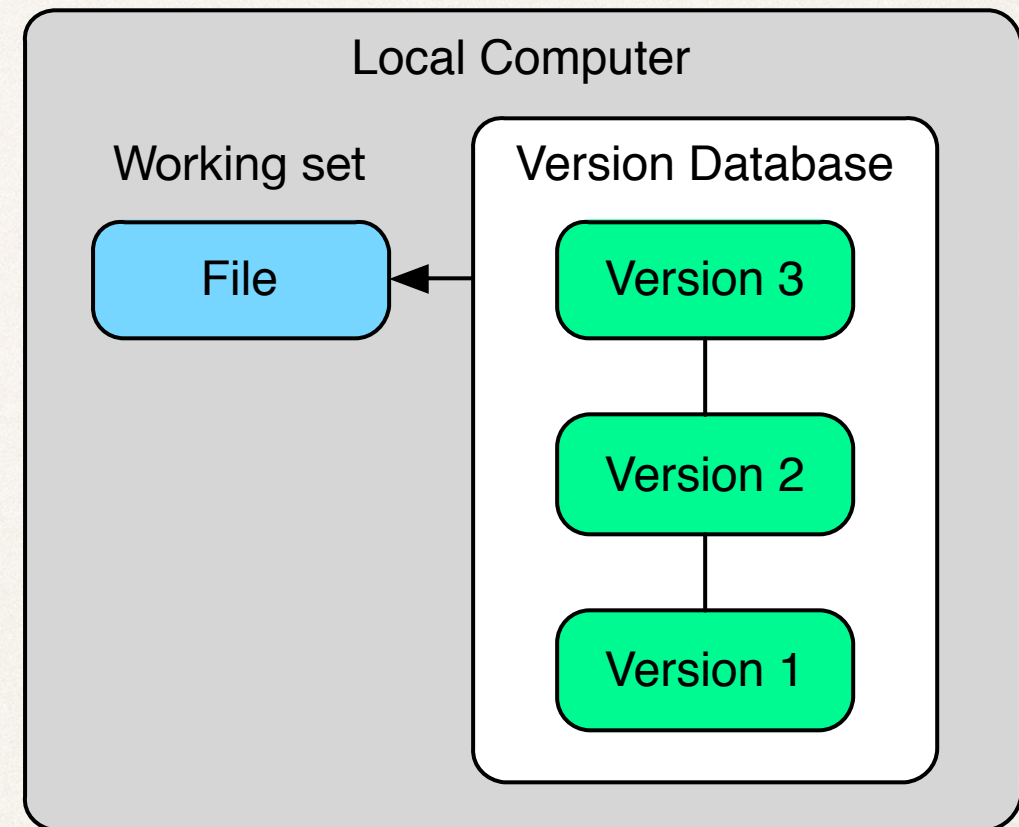
Track changes to directories and files over time

Keeps state apart from working files

Allow previous versions to be recovered

No need for old versions to clutter things up

If mistakes get made 'roll back' to good version





# Distributed collaboration

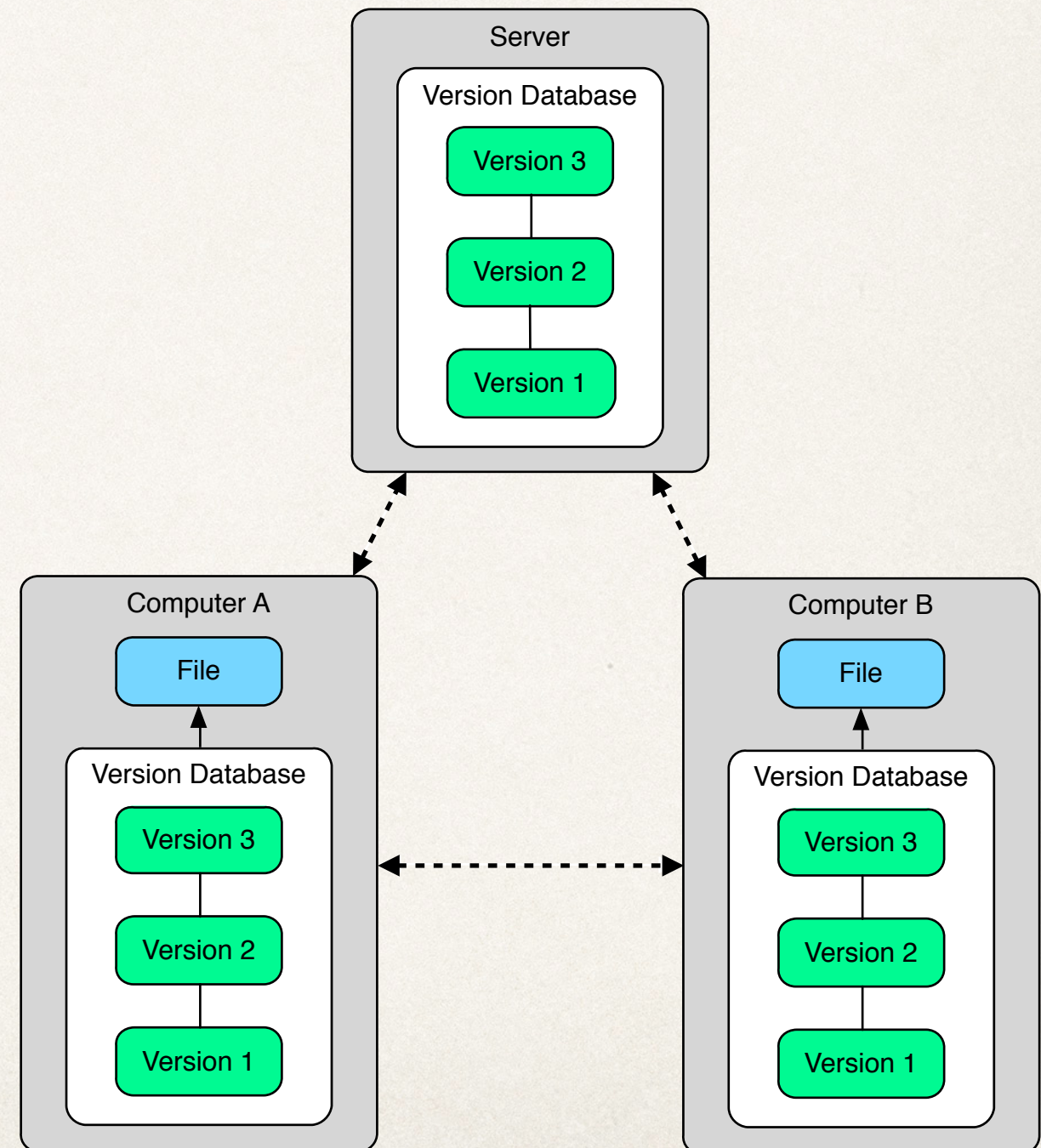
Multiple people in multiple locations with multiple copies

Independent code edits

Share changes

Possibly a central repository

Easy to see who did what when (and roll back)





# Snapshots

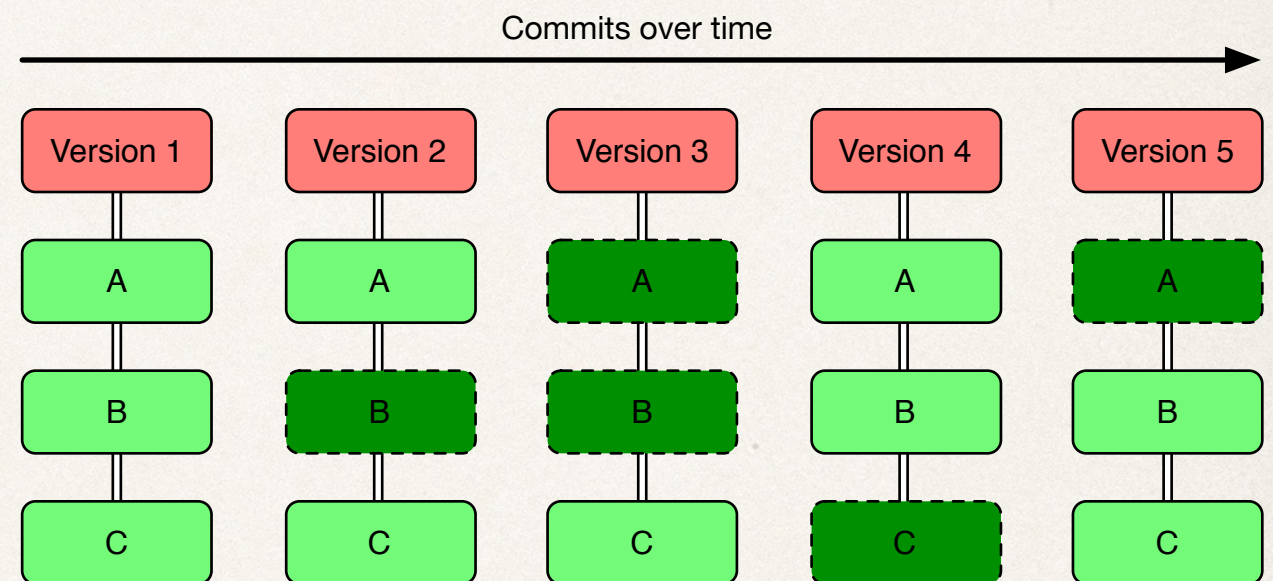
---

Think of the repository as a series of snapshots (saves)

The repository is like a mini file system

Changes to the repository are scheduled using commands like *add*, *remove*, *move* and so on

Snapshots are made using *commit*





# File workflow

---

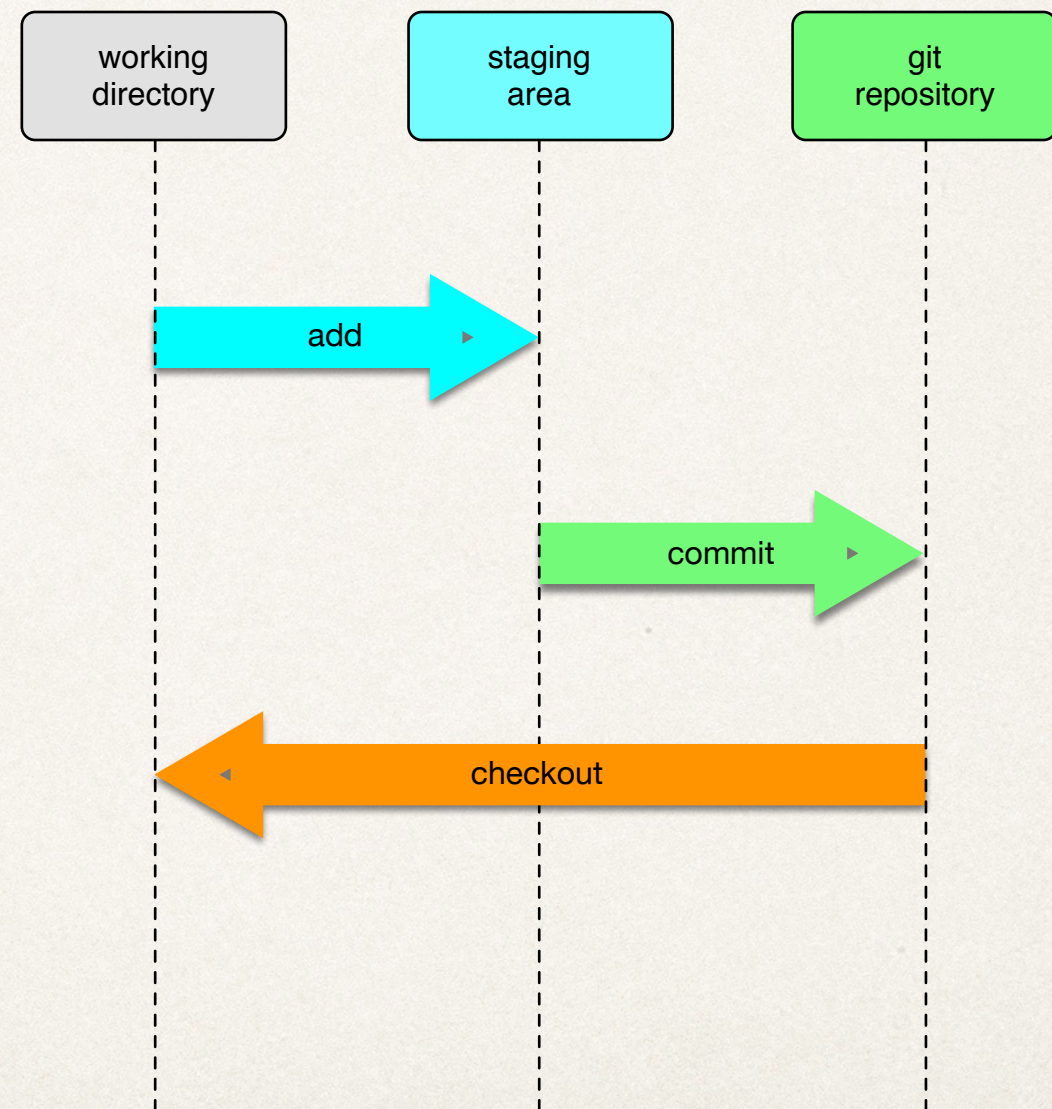
Modify files locally

Stage files for addition, removal, &c.

Perform a *commit* which makes a snapshot out of the staging area

Files can be

- ❖ modified (yet to be staged)
- ❖ staged (yet to be committed)
- ❖ committed





# Removing files

---

We add files for tracking using `git add`, remove them using `git rm`

An IDE with integrated revision control (like RStudio) does this for us

Very important to do it this way rather than just remove the file

Source control is all about keeping track of changes

If you just remove a file, *git* doesn't know you've removed it

The nice thing about having a history though is nothing is gone forever

Tracking changes means telling *git* about the changes

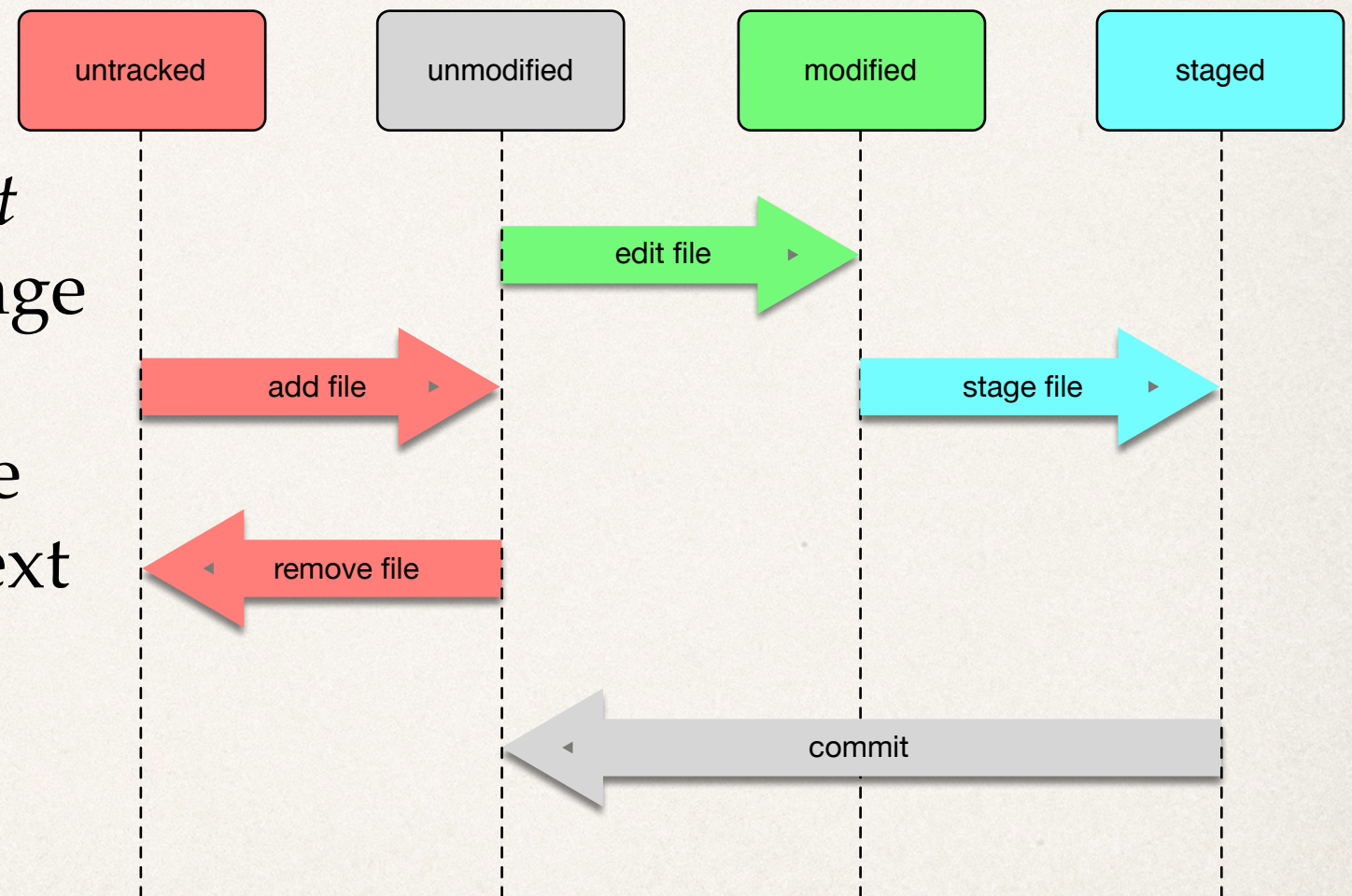


# File status

`git status` shows the status of files

As we *add*, *edit*, *stage* and *commit* files the status of a file will change

During a *commit*, staged files are put into the repository as the next snapshot



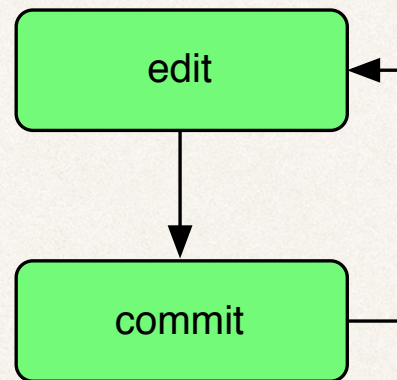


# Routine workflow

---

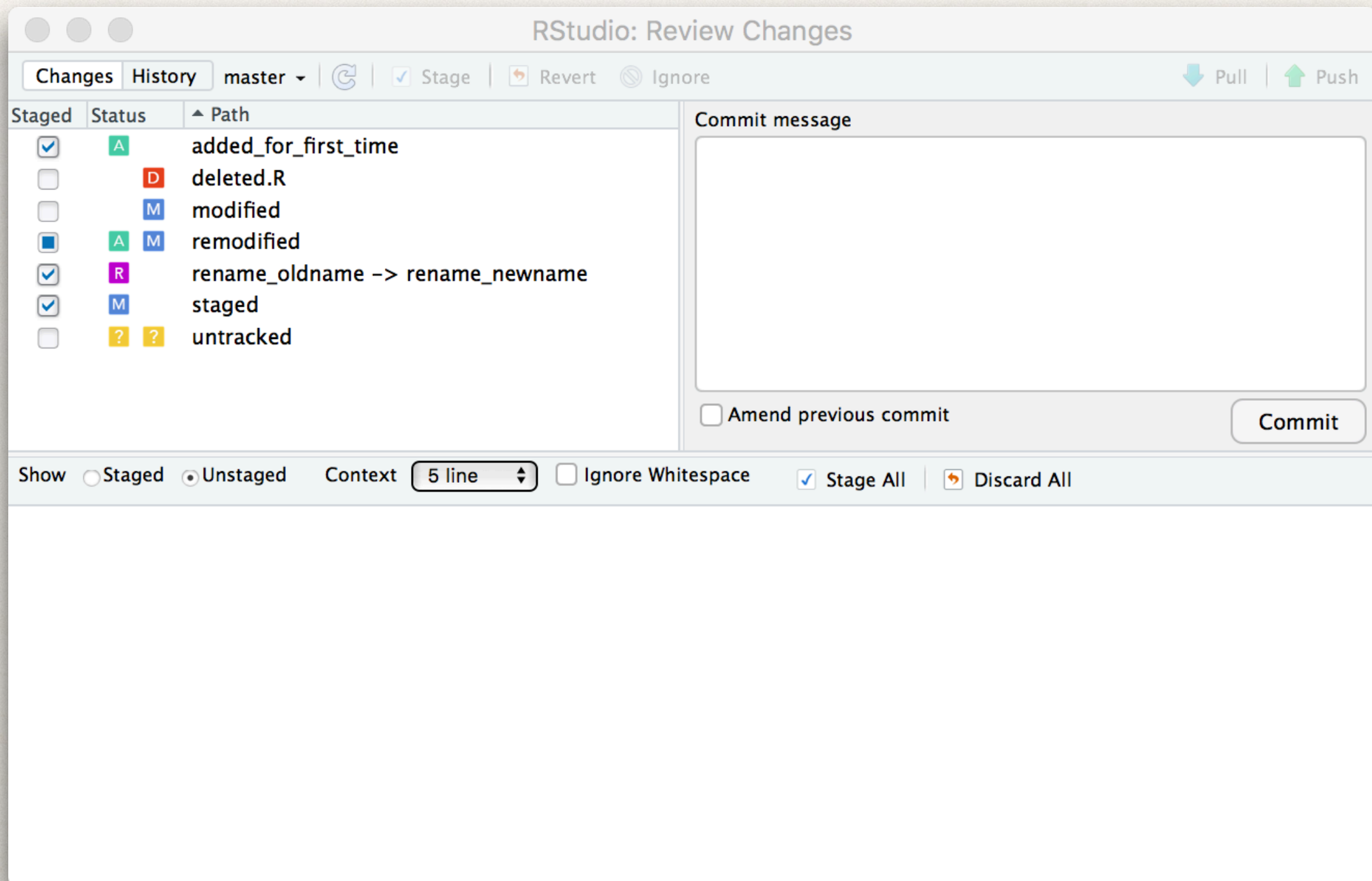
The routine file workflow

Edit → add → commit → edit → add → commit → ...





# Committing in RStudio





# Commit messages

---

When we commit we have to supply a message saying what is in the commit, *i.e.* what changes have been made

## Bad commit messages

- ✦ Fixed bug
- ✦ Committed changes
- ✦ asdifhasodifuhaspdifu

## Good commit message

- ✦ Bump sorting column index by one to fix a problem where the addition of the source study column meant tables were no longer sorted by descending time.



# The commit log: *git log*

---

The `git log` command show us the history of commits

```
$ git log
commit b026324c6904b2a9cb4b88d6d61c81d16d7fce9f (HEAD -> master)
Author: James Farrow <james@fn.com.au>
Date:   Fri Apr 13 18:15:22 2017 +1000
```

[TYPES] Refactor extract.py.

```
commit f3cb6641123c49a8b036b408a548baab47267f03
Author: James Farrow <james@fn.com.au>
Date:   Fri Apr 13 17:59:30 2017 +1000
```

[TYPES] Add extract type facility to extract.py

```
commit 1dcca23355272056f04fe8bf20edfce09b6252c8
Author: James Farrow <james@fn.com.au>
Date:   Fri Apr 13 16:57:46 2017 +1000
```

Better handling of python executable and default\_address.py.



# Each commit has a name

---

Consider it a long id

```
$ git log -n 1 b026324c6904b2a9cb4b88d6d61c81d16d7fce9f
commit b026324c6904b2a9cb4b88d6d61c81d16d7fce9f
Author: James Farrow <james@fn.com.au>
Date:   Fri Apr 13 18:15:22 2017 +1000
```

```
[TYPES] Refactor extract.py.
```

Only a unique prefix is needed so usually you'll only need shorter references to commits like b026324

```
$ git log -n 1 b026324
commit b026324c6904b2a9cb4b88d6d61c81d16d7fce9f
Author: James Farrow <james@fn.com.au>
Date:   Fri Apr 13 18:15:22 2017 +1000
```

```
[TYPES] Refactor extract.py.
```



# Commit hashes are not integers

---

If you've used version control systems before you may be used to a version number of a steadily increasing integer

Because *git* is distributed and disconnected and many people can work on a project at the same time it is not possible to assign such an identifier

They're just used to identify a particular commit



# Commit history in RStudio

RStudio: Review Changes

Changes History master (all commits) Search Pull

| Subject   | Author                         | Date       | SHA      |
|---|--------------------------------|------------|----------|
| HEAD -> refs/heads/master Bumped version.                                     | James Farrow <james@fn.com.au> | 2018-07-19 | 28c83040 |
| origin/master Reformatted DESCRIPTION file.                                   | James Farrow <james@fn.com.au> | 2018-07-18 | 3fb7d6d5 |
| Renamed tutorial directory back to rintro. Added documentation for section(). | James Farrow <james@fn.com.au> | 2018-07-18 | 6dc509fe |
| Rename tutorial directory. Added dependencies.                                | James Farrow <james@fn.com.au> | 2018-07-18 | 4498aa6c |
| Bump version number and only export section() function.                       | James Farrow <james@fn.com.au> | 2018-07-18 | 1a12febb |
| Moved acimcombinedcounts.csv and fixed code.                                  | James Farrow <james@fn.com.au> | 2018-07-18 | a4bbab2d |
| Changed name of tutorial runner to avoid warnings                             | James Farrow <james@fn.com.au> | 2018-07-18 | 0c0bf06c |

Commits 1-43 of 43

SHA 28c83040  
Author James Farrow <james@fn.com.au>  
Date 2018-07-19 03:12  
Subject Bumped version.  
Parent 3fb7d6d5

DESCRIPTION

DESCRIPTION View file @ 28c83040

```
@@ -1,7 +1,7 @@  
1 1 Package: cbdrh.rintro  
2 2 Type: Package  
3 3 Title: Introduction to R Tutorial  
4 4 Version: 1.0.5  
4 4 Version: 1.0.6  
5 5 Authors@R: as.person(c(  
6 6   "James Farrow <james.farrow@unsw.edu.au> [aut, cre]",  
7 7   "Tim Churches <tim.churches@unsw.edu.au> [aut]"
```



# Talking about revisions

---

We can of course refer to revisions by those specific long ids

Git gives us other, more convenient, ways to talk about revisions

HEAD is the current revision of the branch that you're working on

The name of a branch, *e.g.* master can be used to refer to the latest revision on that branch

HEAD is not always the same as the end of the master branch, *i.e.* the most recent main revision but in our case it usually will be for now

All of these references are *local* to a particular repository

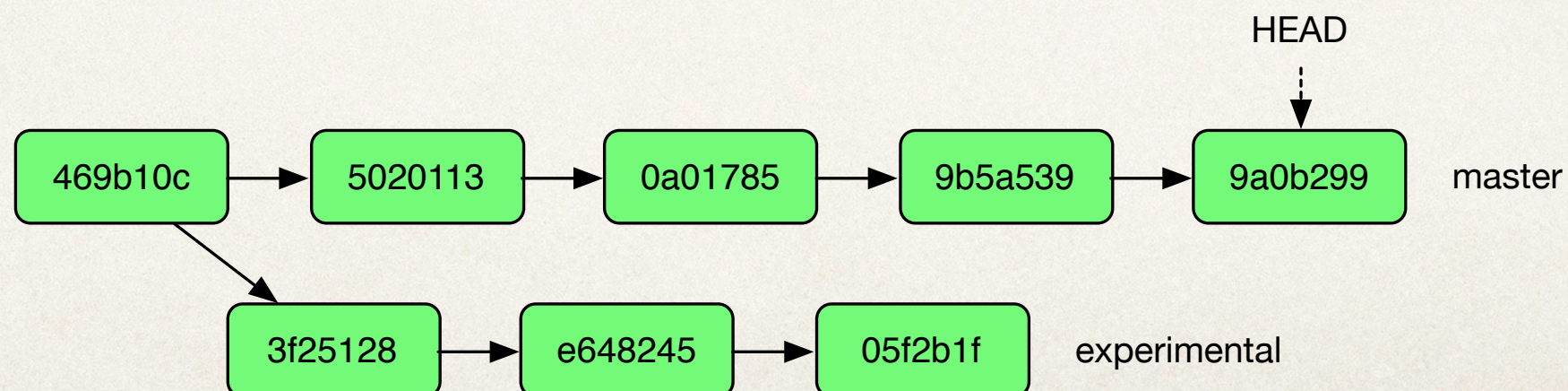


# Branches, revisions and HEAD

---

In the scenario below

- \* *master* would refer to commit 9a0b299
- \* *experimental* would refer to commit 05f2b1f
- \* this repository seems to currently be editing the master branch so *HEAD* would be the same as *master*: 9a0b299





# Good news!

---

You don't really need to remember these details right now

Just rest assured that *git* is more than capable of managing quite complex changes and tracking everything that has gone on

Because *git* has a picture of everything that has happened and how a document went from version A to version B it gives us great control over our documents



# Some useful notations

---

Adding ^ to the end of a reference means 'one commit earlier'

So HEAD^ means 'the revision previous to this one'

While master^ means 'the revision just before the latest on the *master* branch'

Adding @ {...} after a reference allows time-based expressions

- \* master@{yesterday} HEAD@{one month ago}

We can use these to help compare files and see what has changed



# More complex workflows

---

Now we have a history of each file in a project and a way to talk about them we can do more

- ❖ compare different versions of files
- ❖ return to previous working versions
- ❖ safely try out an approach to see if it works and rollback if not
- ❖ have multiple people work on a project concurrently without overwriting each other's files



# Finding differences: *git diff*

---

The *git diff* command shows us what has changed in a file

There are more ways to use *git diff* but here are some important ones

```
git diff [file ...]
```

```
git diff <commit> [--] [file ...]
```

```
git diff <commit> <commit> [--] [file ...]
```

So for example

```
git diff HEAD@{yesterday}
```



# *git diff* in RStudio

RStudio: Review Changes

Changes History master (all commits) Search Pull

|   |                                       |            |          |
|---|---------------------------------------|------------|----------|
| Merge branch 'master' of github.com:CBDRH/Intro-to-R              | James Farrow <james@fn.com.au>        | 2018-07-16 | 64d09a19 |
| Section 6 and moved 'Special values' from section 6 to section 1. | James Farrow <james@fn.com.au>        | 2018-07-16 | 69ec4d80 |
| Fixes to section 1  | Tim Churches <tim.churches@gmail.com> | 2018-07-16 | 9486a00f |
| Added section 5.  | James Farrow <james@fn.com.au>        | 2018-07-13 | 21e089db |
| Incorporated Sabita's feedback on sections 1-3.                   | James Farrow <james@fn.com.au>        | 2018-07-13 | d290a8e5 |
| Added section 4.  | James Farrow <james@fn.com.au>        | 2018-07-13 | 6db6910e |

Commits 1-43 of 43

SHA d290a8e5  
Author James Farrow <james@fn.com.au>  
Date 2018-07-13 05:02  
Subject Incorporated Sabita's feedback on sections 1-3.  
Parent 6db6910e

Intro to R section 1.Rmd  
Intro to R section 2.Rmd  
Intro to R section 3.Rmd

Intro to R section 1.Rmd

|  |   |
|--|---|
| @@ -27,7 +27,7 @@ This section deals with                              |   |
| 27   | 27 * Variable types   |
| 28   | 28 * Basic arithmetic operators: `+`, `-`, `*`, `/`, `^`          |
| 29   | 29 * Basic comparison operators: `<`, `<=`, `>`, `>=`, `==`, `!=` |
| 30   | 30 * Basic logical operations: `&`, ` `, `^`, `!`                 |
| 30   | 30 * Basic logical operations: `&`, ` `, `^`, `!`                 |
| 31   | 31  |
| 32   | 32 ## What is R?  |
| 33   | 33  |
| @@ -38,7 +38,7 @@ R is a high-level, open-source programming language. |   |



# Reverting: *git checkout*

---

The *git checkout* command lets us pull specific files out of the repository

```
git checkout <commit> file ...
```

- ❖ this pulls a specific version of a file out of the repository and *replaces* the current version of the file
- ❖ **any uncommitted work will be lost: if in doubt commit first**

If, for example, we found that we had completely damaged a file and wanted to go back to the way it was yesterday and throw away all our changes we could run

```
git checkout "HEAD@{yesterday}" helpers.R
```



# Reverting: *git checkout*

---

The *git checkout* command lets us pull specific files out of the repository

```
git checkout <commit> file ...
```

- ✧ this pulls a specific version of a file out of the repository and *replaces* the current version of the file
- ✧ **any uncommitted work will be lost: if in doubt commit first**

If, for example, we found that we had completely damaged a file and wanted to go back to the way it was yesterday and throw away all our changes we could run

```
git checkout "HEAD@{yesterday}" helpers.R
```



# Reverting and commit size

---

Being able to revert changes makes revision control very powerful

Of course it needs to be coupled with sensible commit sizes

Too big and too many changes are lumped together

Too small and it's just inconvenient and changes are spread over multiple commits



# Branches

---

Repositories can contain ‘experimental’ versions of a project

Sometimes we want to ‘try something out’

Rather than disrupt everyone we use a *branch*

Work usually occurs on the *master* branch



# Branching

---

Branches split off from the default *master* copy

Weird and wonderful experimentation take place

Work can be folded back into *master*, or...

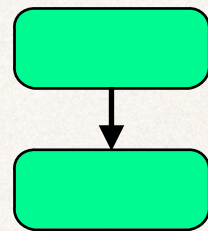
...work can be ignored and abandoned (but not forgotten)

The *master* is the branch everyone uses initially

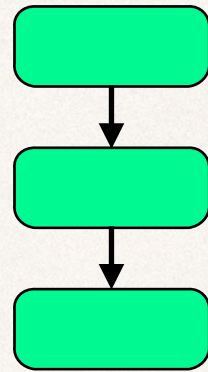




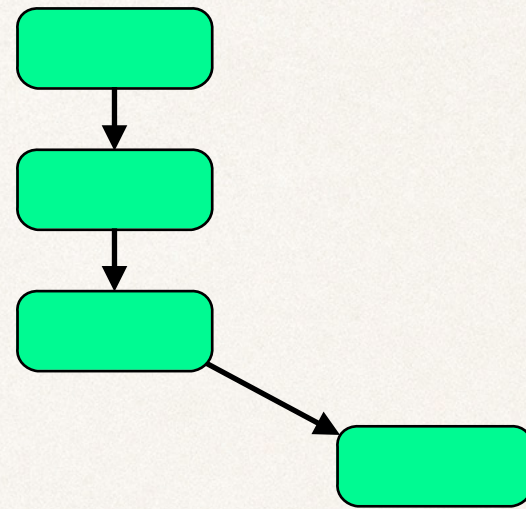




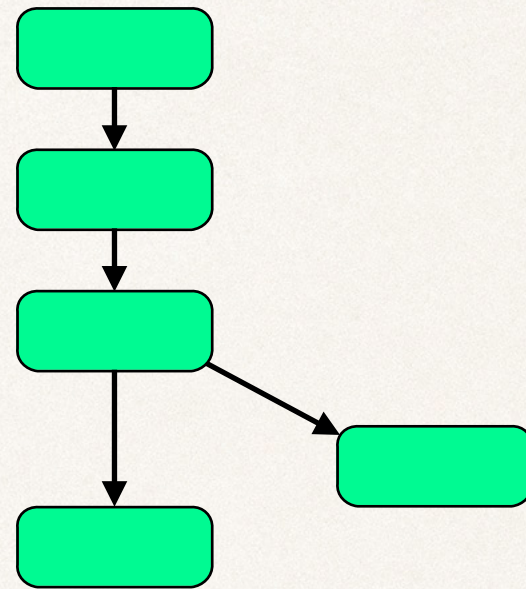




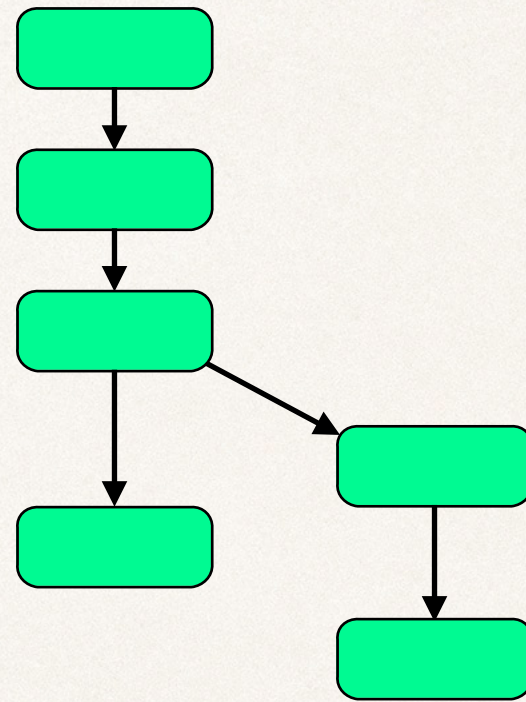




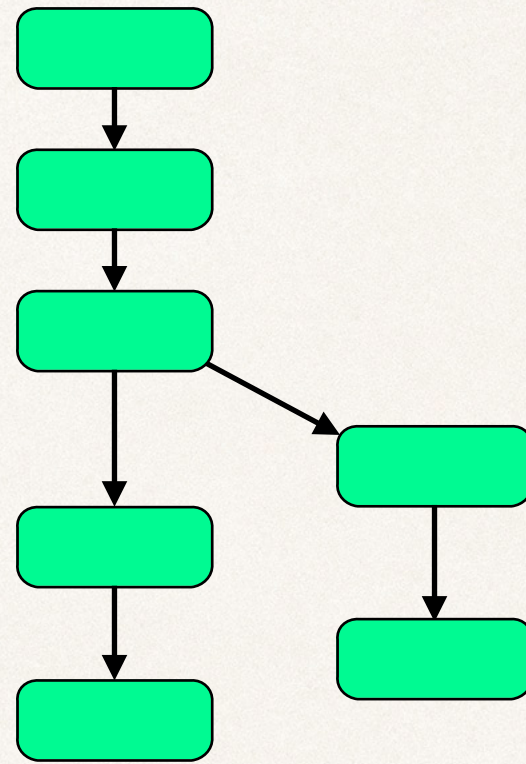




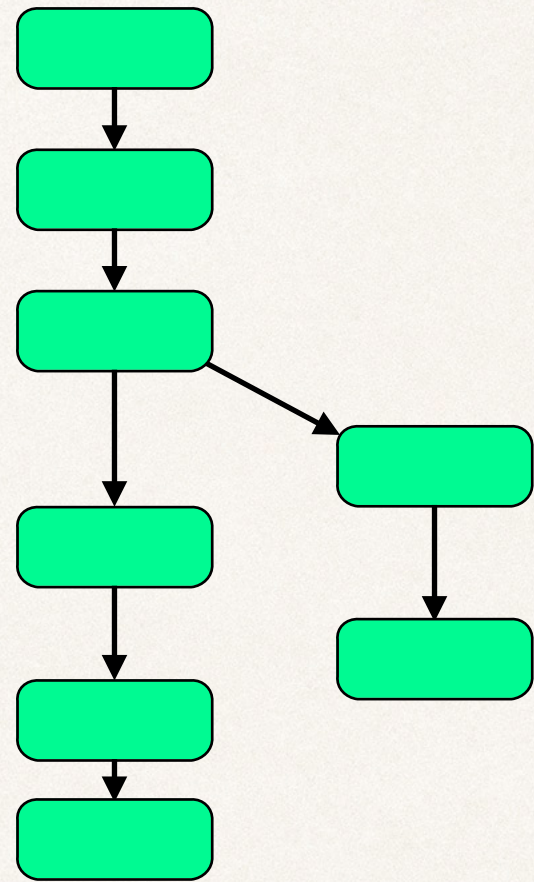




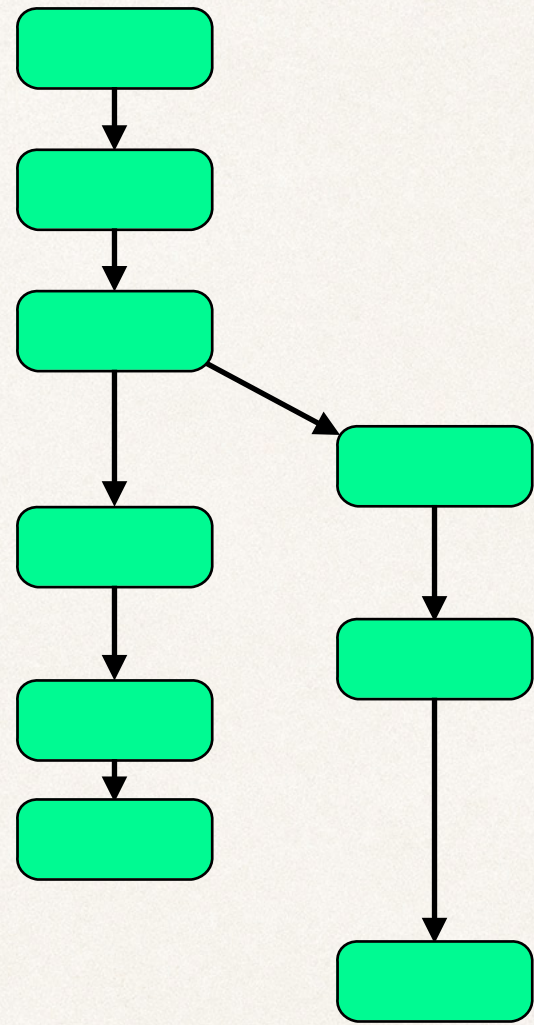




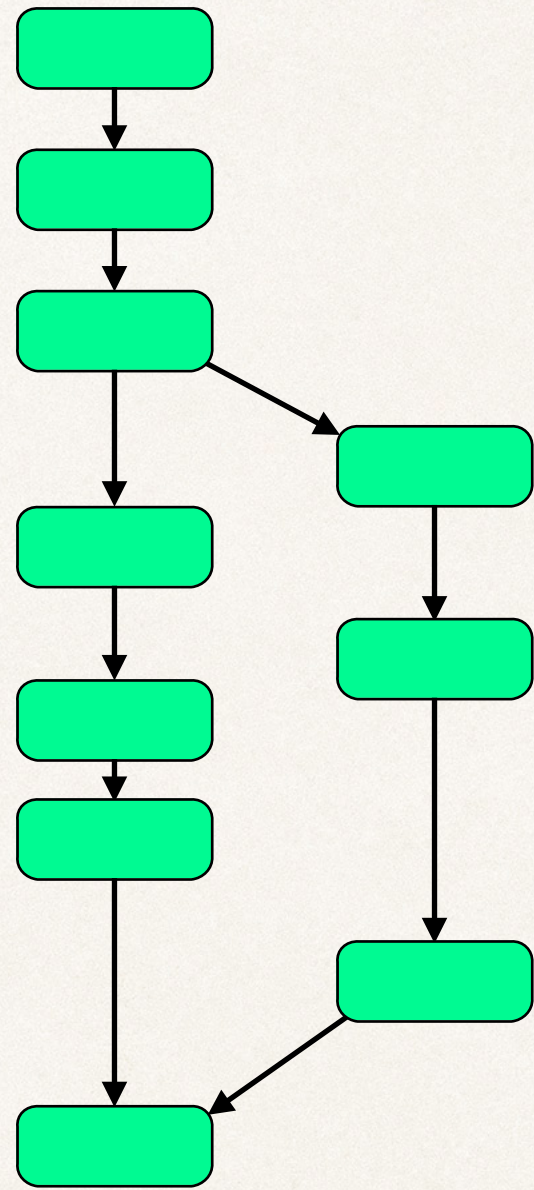




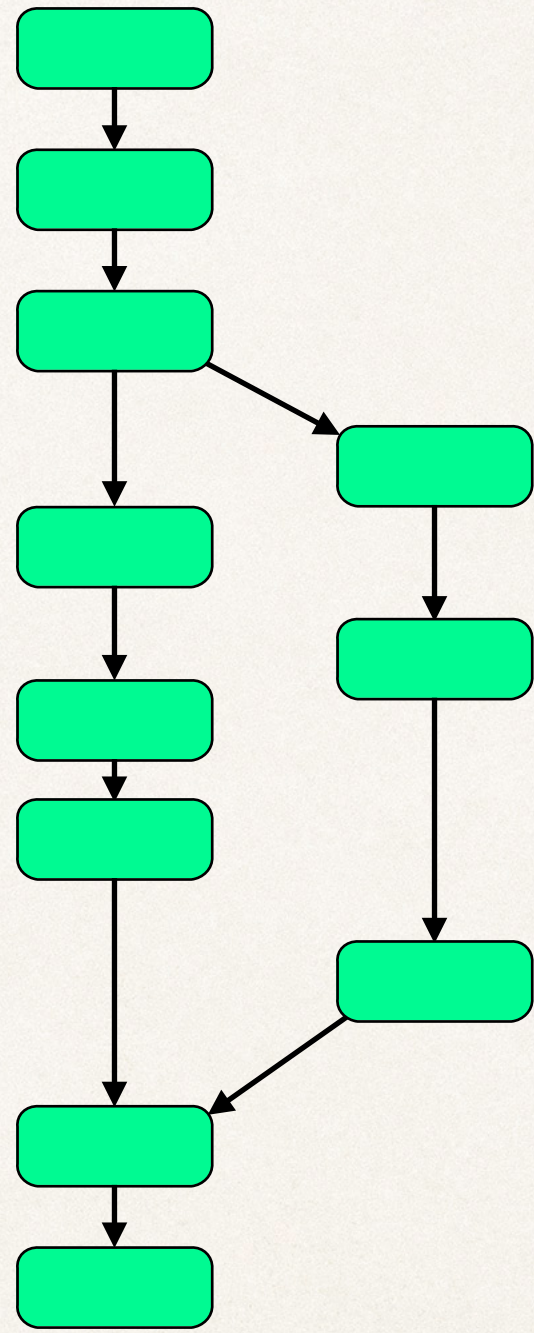




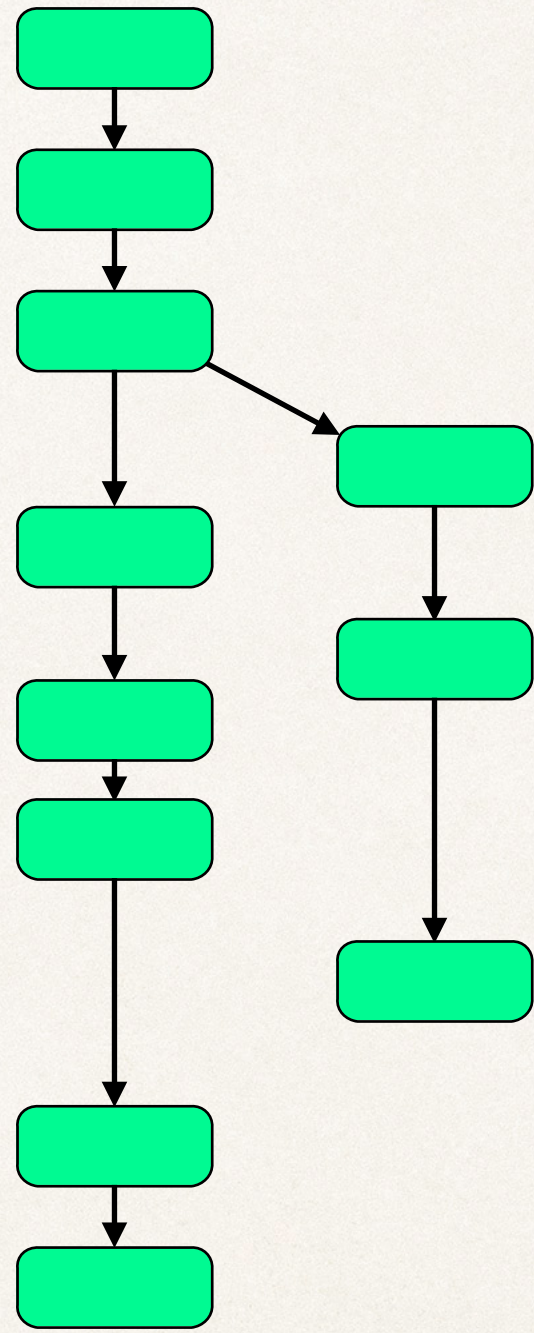














# Branches

---

Branches have the same structure as the trunk from which they are spawned

In fact, at the point of branching they are identical

This is not taking a copy of a project and calling it *my\_project\_v2*

This is having two versions which you can swap back and forth between at will

Keeping the same structure means it's easier to compare the trunk and the branch and see exactly what has changed



# Branching

---

We're not going to look at branching today but be aware it exists

It's a powerful mechanism to make major changes to a project while isolating those changes so that they don't affect everyone else

It's also a good mechanism to make major changes and keep them separate from your own work on the mainline project development

Instead of working for days and not saving 'because you don't want to overwrite the original' or saving a copy as *my\_prog.experimental.R* 'to keep it separate' we can continue to work on *my\_prog.R* and continue to exploit the benefits of regular commits



# Remote repositories

---

So far we have just talked about working within a local repository

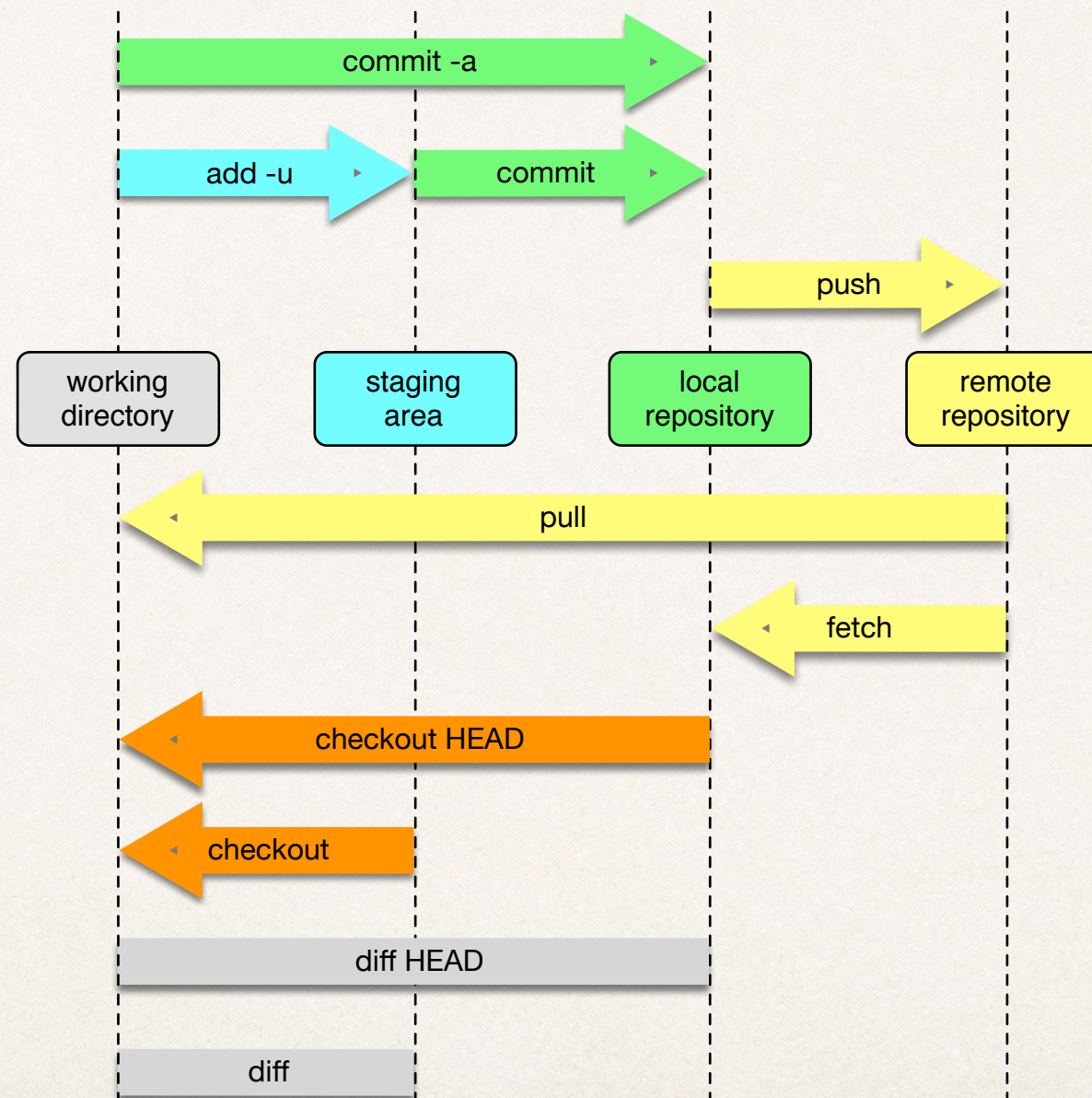
Repositories can also ‘push’ and ‘pull’ changes between repositories

We call repositories that aren’t the local repository *remote* repositories

Remote repositories are important for collaboration



# File workflow revisited





# Conflicts in RStudio

The screenshot shows the RStudio interface with a file conflict in the 'conflict.R' file. The main editor displays the following code:

```
1 <==== HEAD
2 cube <- function(z) {
3   z^3
4   =====
5 cube <- function(n) {
6   n^3
7   >>>>>> f67a67c3cfd0a73f33cdc34f36266c96884bc7cb
8 }
```

The right-hand pane shows the 'Environment' tab with a 'conflict' section. It lists the file 'conflict.R' with a status of 'U' (Unmerged) and a path of 'private/tmp/conflict'.

The bottom-right pane shows the 'Files' tab with a list of files in the 'private/tmp/conflict' directory:

| Name           | Size  | Modified  |
|----------------|-------|-----------|
| ..             |       |           |
| .gitignore     | 40 B  | Jul 30, 2 |
| .Rhistory      | 0 B   | Jul 30, 2 |
| conflict.R     | 127 B | Jul 30, 2 |
| conflict.Rproj | 205 B | Jul 30, 2 |

The bottom-left pane shows the console output, which includes the R startup message and the R version information:

```
Platform: x86_64-apple-darwin17.6.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

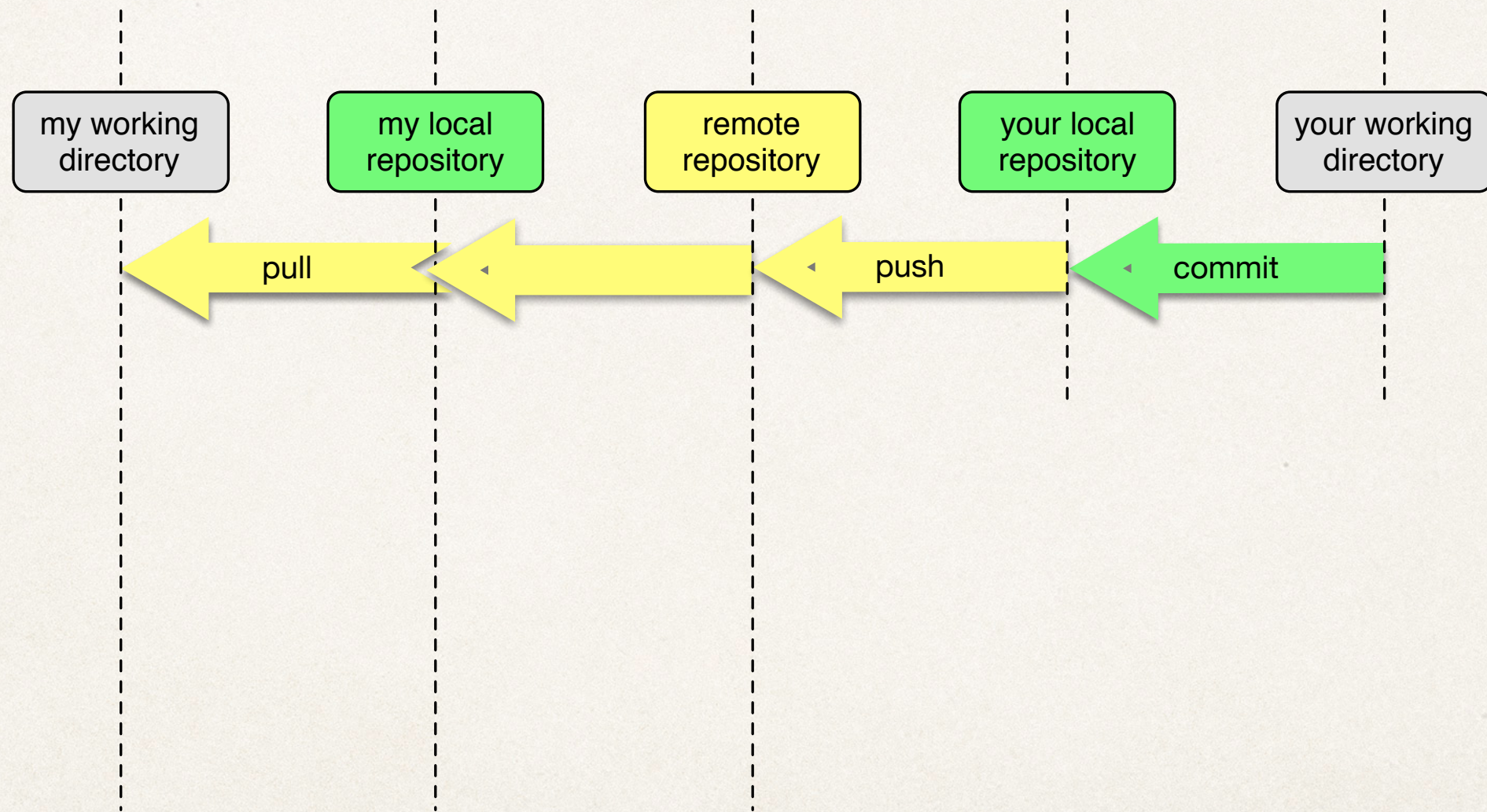
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```



# Collaborative workflow

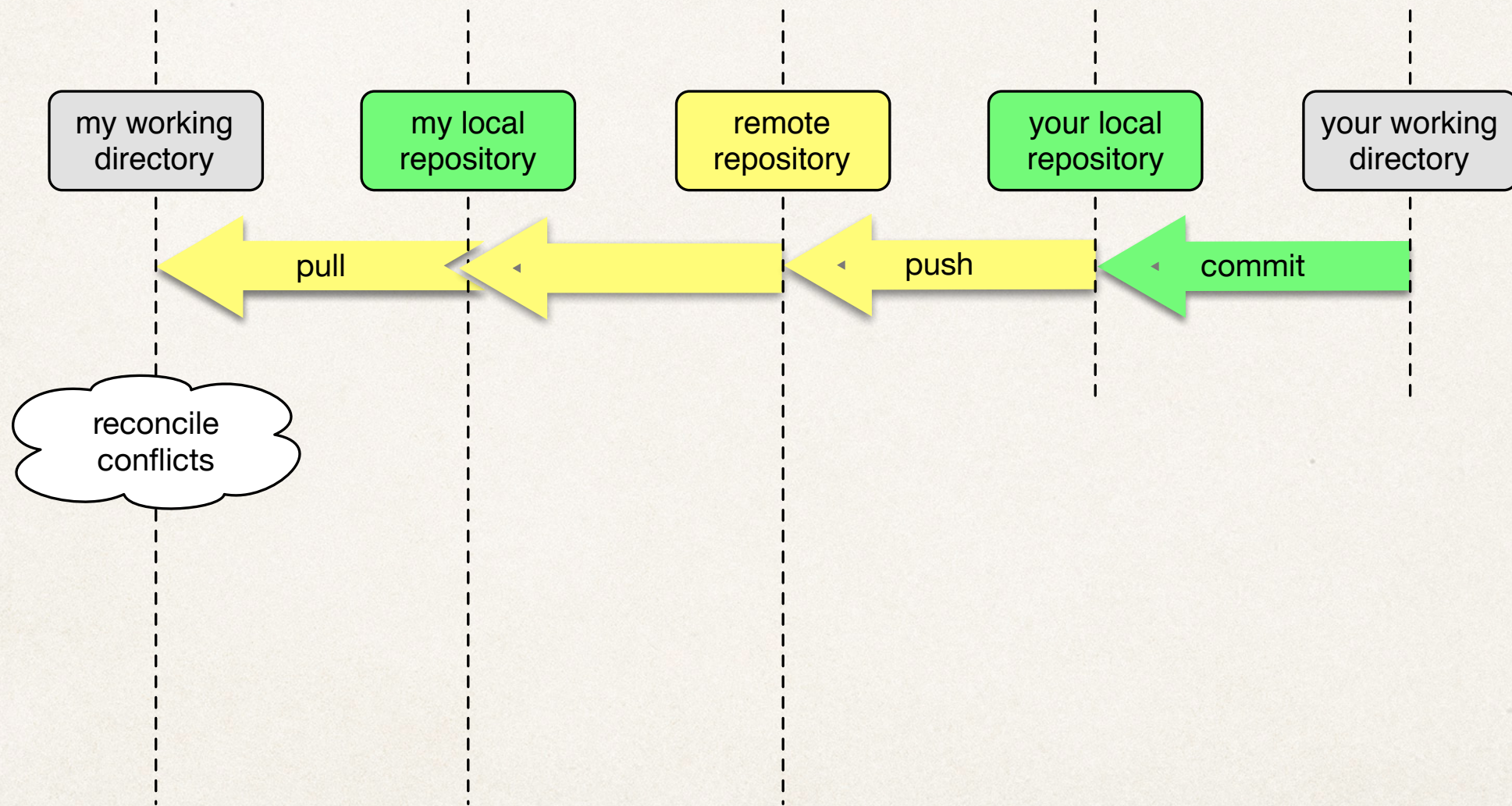
---





# Collaborative workflow

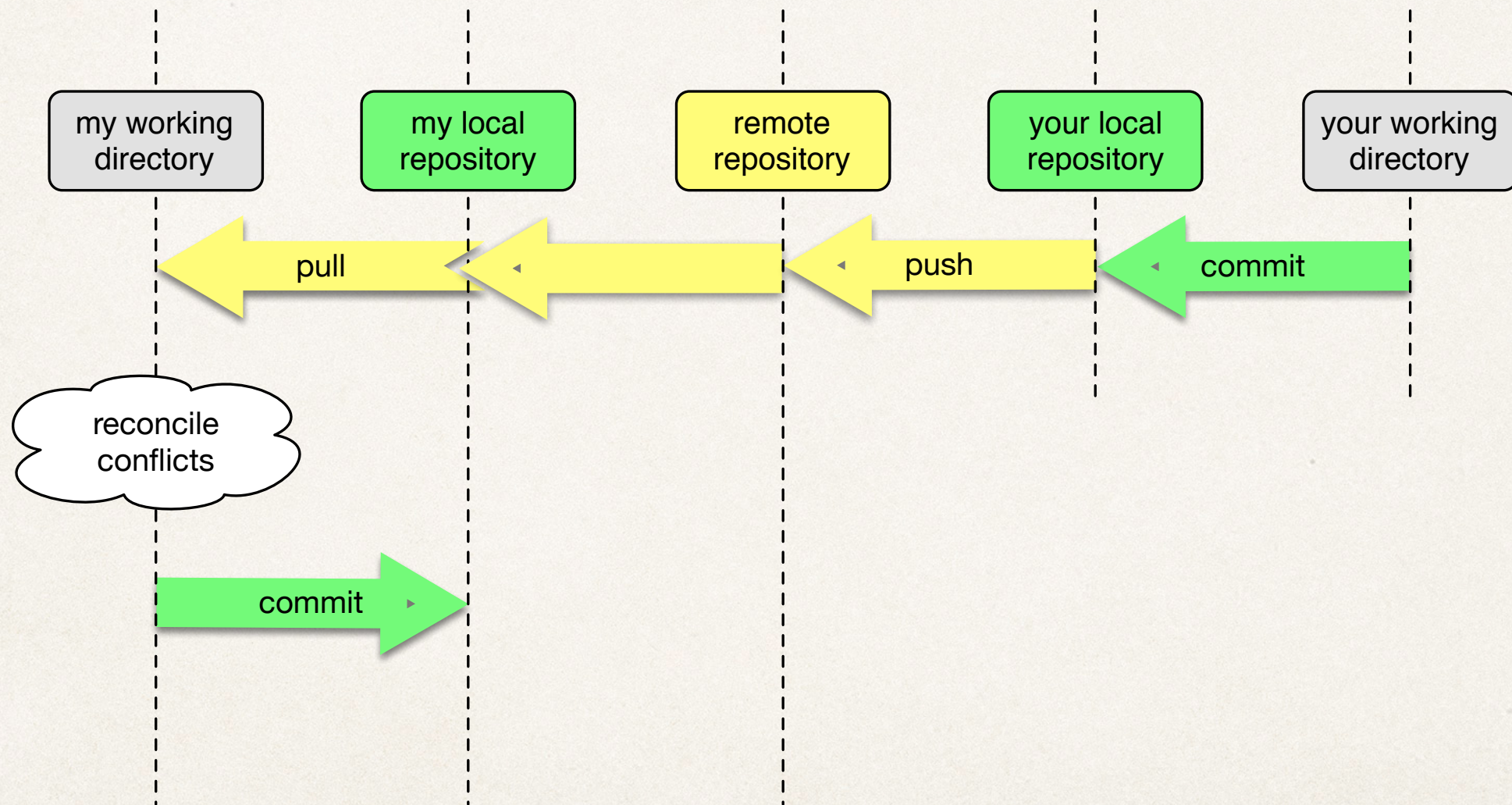
---





# Collaborative workflow

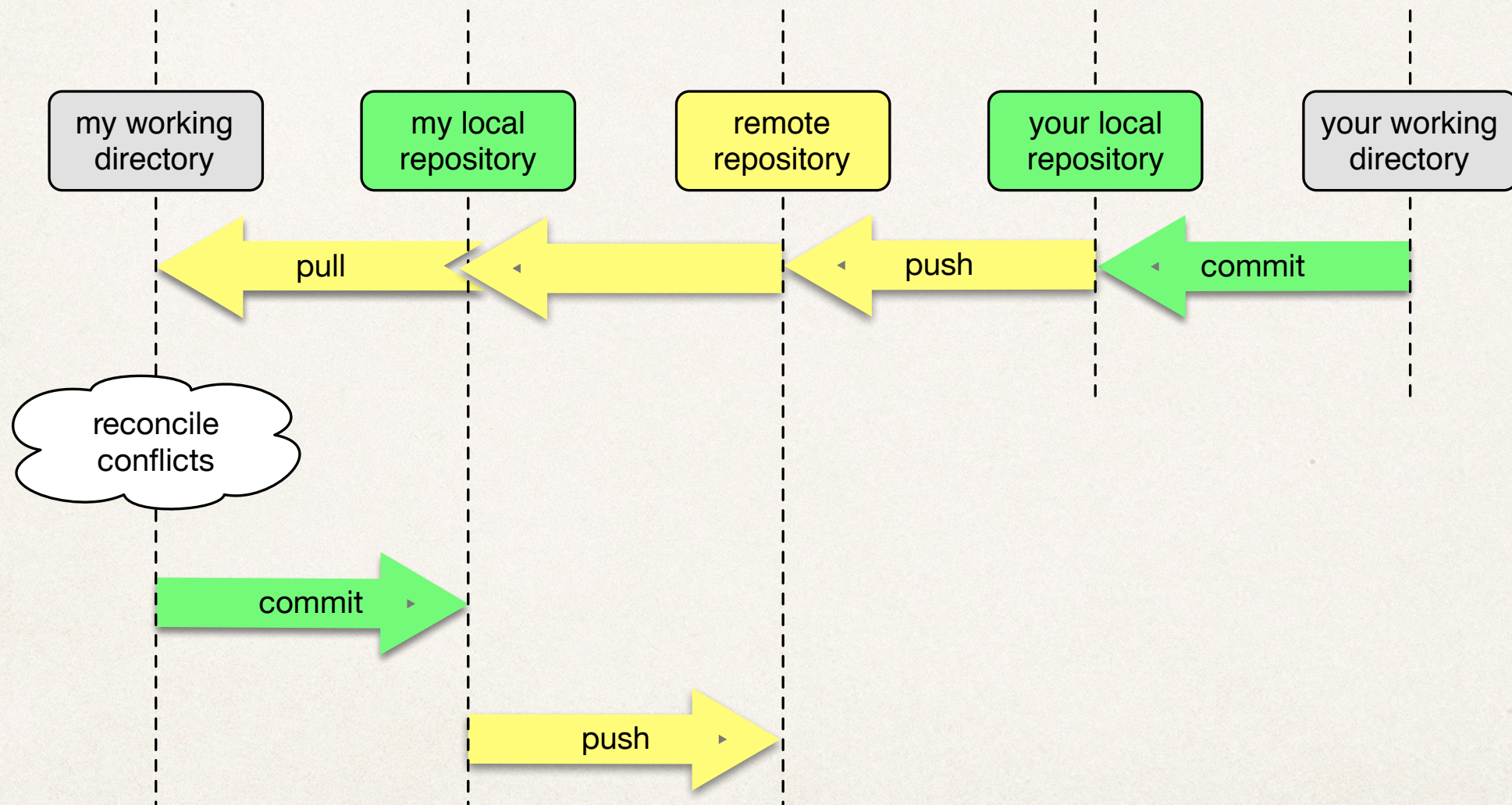
---





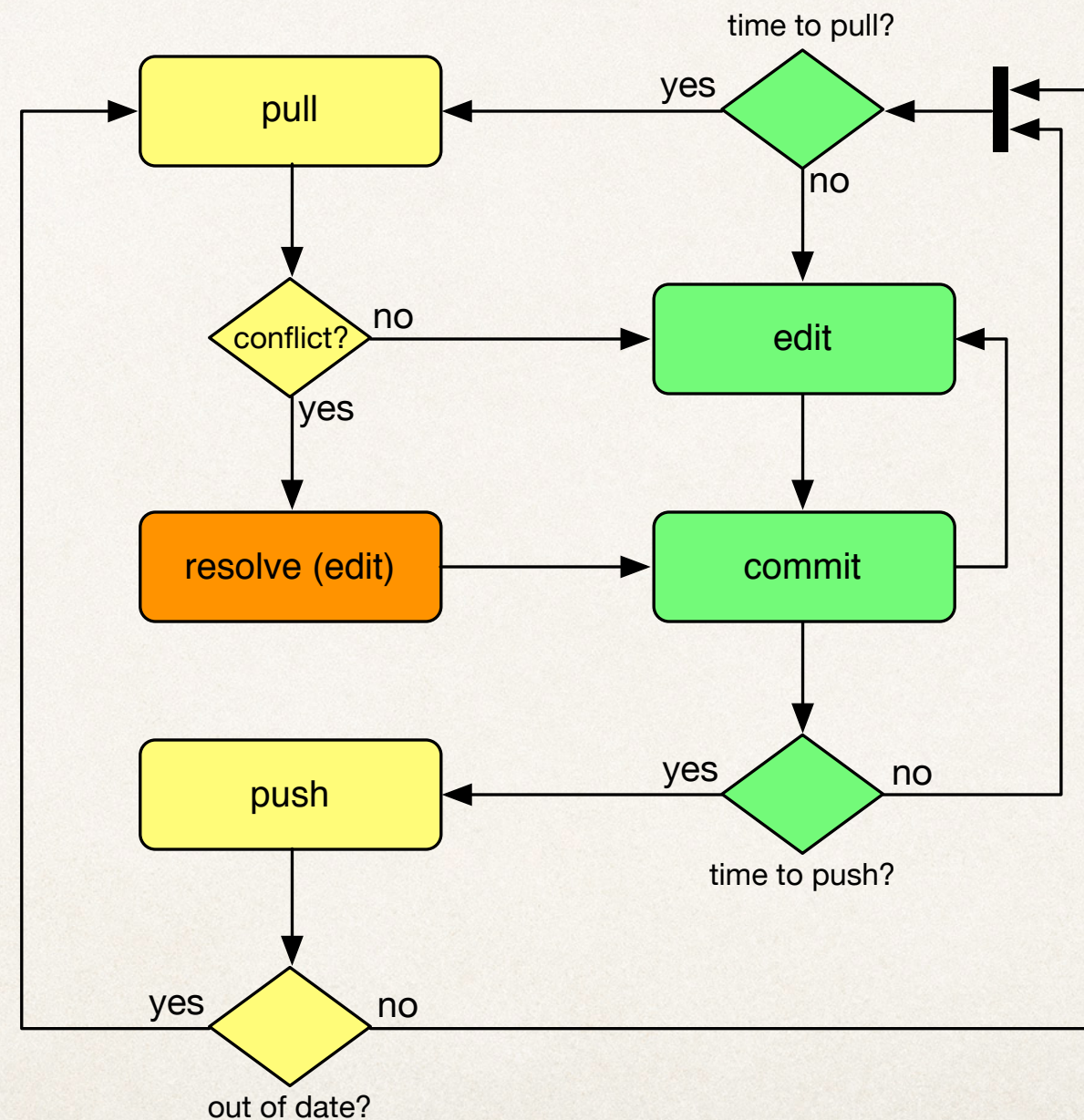
# Collaborative workflow

---



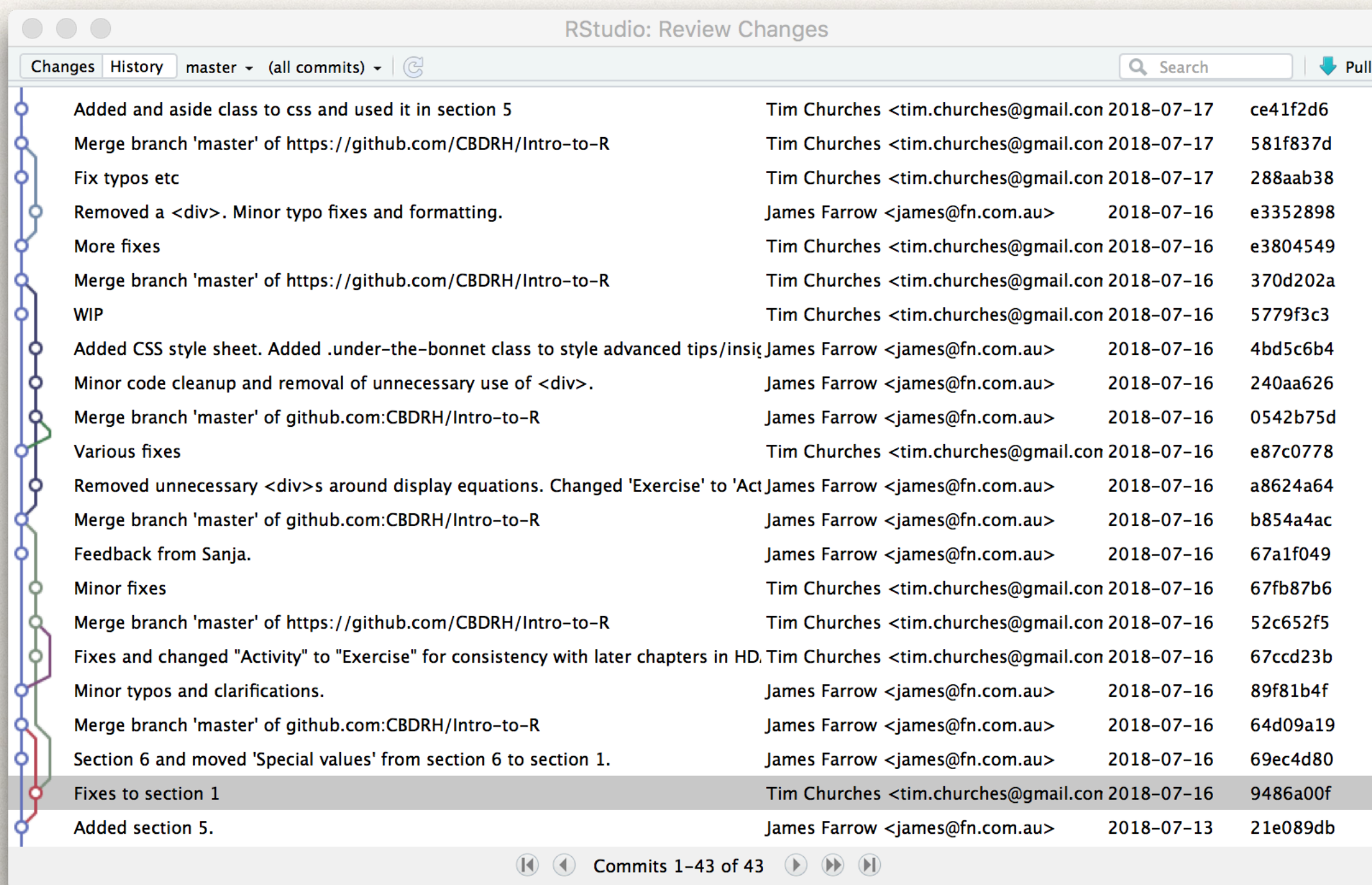


# Collaborative workflow





# Commit history in RStudio



The screenshot shows the 'RStudio: Review Changes' window. At the top, there are tabs for 'Changes' and 'History', with 'History' selected. Below the tabs, there's a dropdown menu showing 'master' and '(all commits)'. To the right of the dropdown is a search bar labeled 'Search' and a 'Pull' button with a downward arrow. The main area displays a list of commits, each with a description, author, date, and commit hash. The commits are listed in reverse chronological order. The first commit is 'Added and aside class to css and used it in section 5' by Tim Churches, dated 2018-07-17, with hash ce41f2d6. The last commit is 'Added section 5.' by James Farrow, dated 2018-07-13, with hash 21e089db. The window also features a commit graph on the left side, showing the progression of branches and commits.

| Commit Description   | Author                                | Date       | Commit Hash |
|--|---------------------------------------|------------|-------------|
| Added and aside class to css and used it in section 5                                | Tim Churches <tim.churches@gmail.com> | 2018-07-17 | ce41f2d6    |
| Merge branch 'master' of https://github.com/CBDRH/Intro-to-R                         | Tim Churches <tim.churches@gmail.com> | 2018-07-17 | 581f837d    |
| Fix typos etc  | Tim Churches <tim.churches@gmail.com> | 2018-07-17 | 288aab38    |
| Removed a <div>. Minor typo fixes and formatting.                                    | James Farrow <james@fn.com.au>        | 2018-07-16 | e3352898    |
| More fixes   | Tim Churches <tim.churches@gmail.com> | 2018-07-16 | e3804549    |
| Merge branch 'master' of https://github.com/CBDRH/Intro-to-R                         | Tim Churches <tim.churches@gmail.com> | 2018-07-16 | 370d202a    |
| WIP  | Tim Churches <tim.churches@gmail.com> | 2018-07-16 | 5779f3c3    |
| Added CSS style sheet. Added .under-the-bonnet class to style advanced tips/insig    | James Farrow <james@fn.com.au>        | 2018-07-16 | 4bd5c6b4    |
| Minor code cleanup and removal of unnecessary use of <div>.                          | James Farrow <james@fn.com.au>        | 2018-07-16 | 240aa626    |
| Merge branch 'master' of github.com:CBDRH/Intro-to-R                                 | James Farrow <james@fn.com.au>        | 2018-07-16 | 0542b75d    |
| Various fixes  | Tim Churches <tim.churches@gmail.com> | 2018-07-16 | e87c0778    |
| Removed unnecessary <div>s around display equations. Changed 'Exercise' to 'Act      | James Farrow <james@fn.com.au>        | 2018-07-16 | a8624a64    |
| Merge branch 'master' of github.com:CBDRH/Intro-to-R                                 | James Farrow <james@fn.com.au>        | 2018-07-16 | b854a4ac    |
| Feedback from Sanja.   | James Farrow <james@fn.com.au>        | 2018-07-16 | 67a1f049    |
| Minor fixes  | Tim Churches <tim.churches@gmail.com> | 2018-07-16 | 67fb87b6    |
| Merge branch 'master' of https://github.com/CBDRH/Intro-to-R                         | Tim Churches <tim.churches@gmail.com> | 2018-07-16 | 52c652f5    |
| Fixes and changed "Activity" to "Exercise" for consistency with later chapters in HD | Tim Churches <tim.churches@gmail.com> | 2018-07-16 | 67ccd23b    |
| Minor typos and clarifications.  | James Farrow <james@fn.com.au>        | 2018-07-16 | 89f81b4f    |
| Merge branch 'master' of github.com:CBDRH/Intro-to-R                                 | James Farrow <james@fn.com.au>        | 2018-07-16 | 64d09a19    |
| Section 6 and moved 'Special values' from section 6 to section 1.                    | James Farrow <james@fn.com.au>        | 2018-07-16 | 69ec4d80    |
| Fixes to section 1   | Tim Churches <tim.churches@gmail.com> | 2018-07-16 | 9486a00f    |
| Added section 5.   | James Farrow <james@fn.com.au>        | 2018-07-13 | 21e089db    |

Commits 1-43 of 43



# Setting up *git*

---

Install *git*

Windows

- ❖ download from <https://git-scm.com/download/win>

MacOS

- ❖ install Xcode tools: `xcode-select --install`
- ❖ install via Homebrew: <https://brew.sh>

Linux users should use their package manager



# Installing *git* on Windows

---

Choose an editor that you're happy using: it only needs to edit text files

For Windows 'Use git from the the Window Command Prompt' is the best option unless you need something different

Use OpenSSL unless you have corporate requirements

For line endings, choose the first (`core.autocrlf = true`) option

Use *minTTY* for terminal emulation

Basically, leave things at their *default values*



# Configuring *git*

---

You need to let *git* know who you are

This can be done *globally* (for all local repositories) and overridden in each local repository as necessary

This enables *git* to record *authorship* information: who did what

You want your contributions to be recognised, yes?

- ❖ especially when it comes to assignments 😊



# Setting up GitHub

---

GitHub is not *git*

GitHub is a site which manages a git repository

We will be using GitHub for assignments and in workshops

Go to *<https://github.com>* and create an account if you don't have one

You don't need to use your uni/work email address

GitHub can be configured to recognise many addresses as 'you'



# Configuring *git*

---

Open a terminal window

- ❖ The *Terminal* app under macOS can be found using *Spotlight* or opened from the *Utilities* folder opened from the *Go* menu in the *Finder*
- ❖ Open *Git Bash* under Windows using the *Start* menu

Type the following (replacing the details between the "...")

```
git config --global user.name "James Farrow"
```

```
git config --global user.email "james.farrow@unsw.edu.au"
```



# Configuration

---

You may also be using *git* and *GitHub Classroom* in other courses

Set up your GitHub account and configure *git* with the same identity



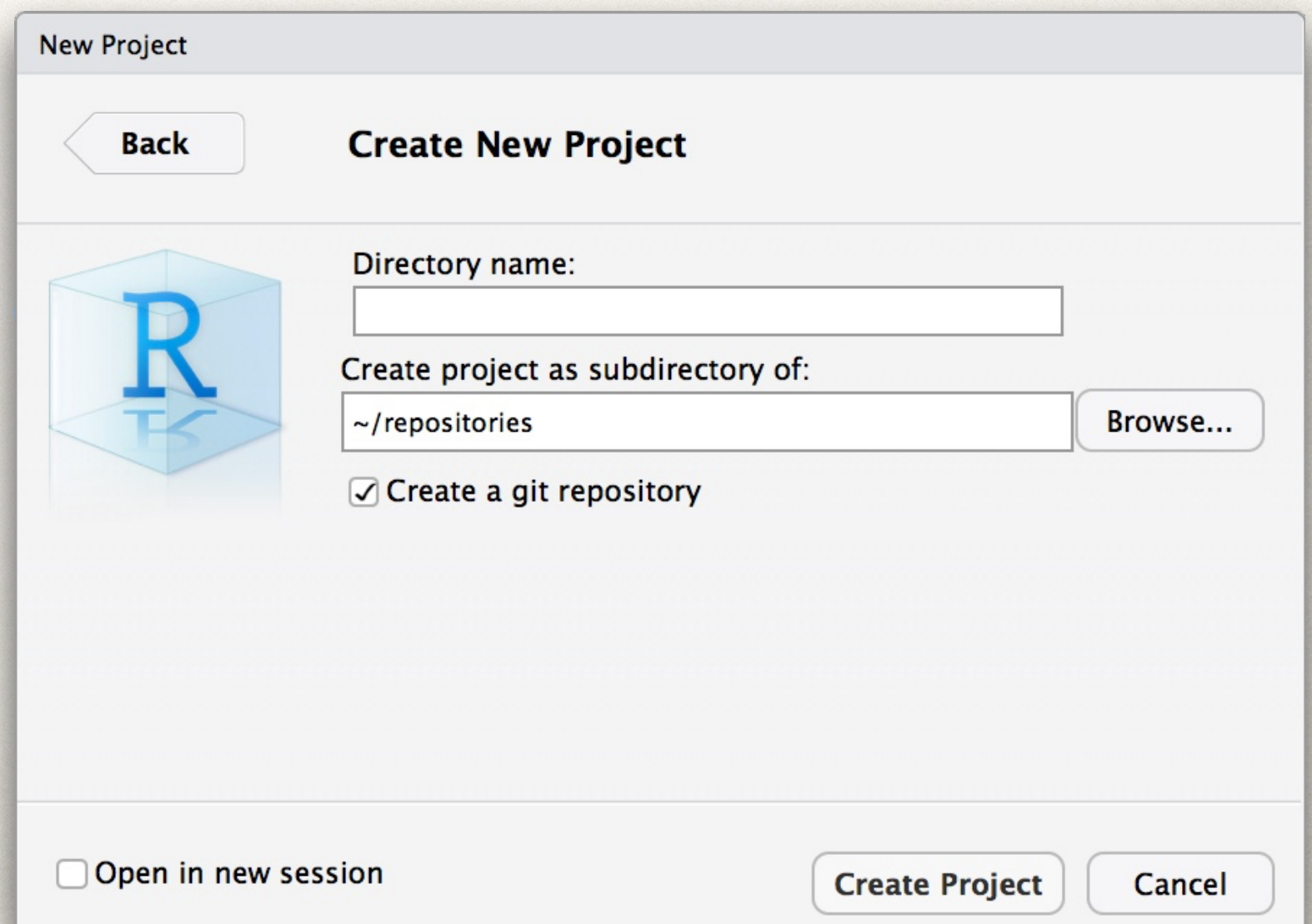
# Setting up RStudio to use *git*

---

When *git* is installed RStudio can be configured to use it for a project

*Always do this*

There is a checkbox on the 'New project' dialog to enable *git*





# Adding *git* to an existing project

---

Run `git init` in the top level directory

This will create a new *.git* directory with an empty local repository

- ✧ Never touch anything in this folder

Use `git add` and `git commit` to add in the existing file

From then on it's business as usual

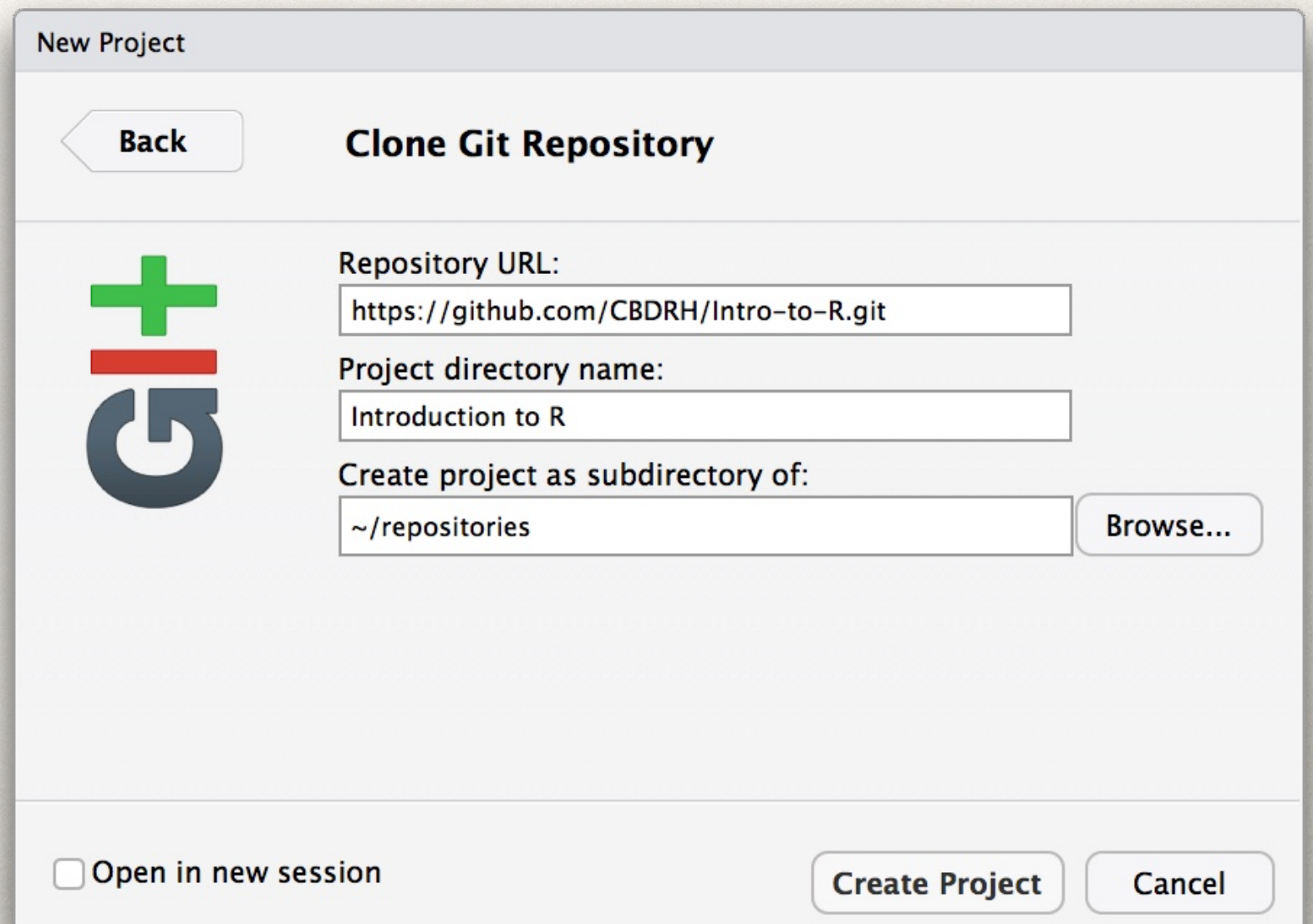


# Cloning a project from *GitHub*

A project repository may already exist on GitHub

To clone this start a new project in RStudio but choose *Version Control* rather than *New Directory* in the initial dialogue box

Then choose *Git* to clone an existing project



The image shows the 'New Project' dialog box in RStudio, specifically the 'Clone Git Repository' tab. The dialog has a title bar 'New Project' and a 'Back' button. On the left is a large Git logo (a green plus sign over a blue 'G'). The main area contains three text input fields: 'Repository URL:' with the value 'https://github.com/CBDRH/Intro-to-R.git', 'Project directory name:' with the value 'Introduction to R', and 'Create project as subdirectory of:' with the value '~/repositories'. A 'Browse...' button is next to the last field. At the bottom, there is a checkbox 'Open in new session' and two buttons: 'Create Project' and 'Cancel'.

New Project

Back

**Clone Git Repository**

Repository URL:  
https://github.com/CBDRH/Intro-to-R.git

Project directory name:  
Introduction to R

Create project as subdirectory of:  
~/repositories Browse...

☐ Open in new session

Create Project Cancel



# Using GitHub Classroom

---

We will be doing weekly assessments (and the group assignment) using *GitHub Classroom*

Each week you will receive a link for that week's assessment

You *must* have a GitHub account to participate in the assessments

Following the link will create a new repository for you for each assessment with the start files for that assessment already set up

You need to *clone* each repository, *commit* your work and *push* the results

Whatever is in the repository at each deadline is what will be marked



# Assessment repositories

---

Each assessment will have a name like *Chapter 1 Assessment*

When you join each assessment exercise *GitHub Classroom* will create a new repository for you with your *GitHub* username appended

CBDRH-HDAT9800 is the organisation

Your repository for Chapter 1 Assessment will be something like

- ❖ *<https://github.com/CBDRH-HDAT9800/chapter-1-assessment-jmfarrow>*

*GitHub Classroom* will send you links to all of this, you don't need to remember it



# Cloning the assessment repository

The screenshot shows a web browser window displaying a GitHub repository page. The address bar shows the URL: `github.com/CBDRH-HDAT9800/chapter-1-assessment-jmfarrow`. The repository is titled "chapter-1-assessment-jmfarrow" and is marked as "Private". It was created by "GitHub Classroom". The repository has 1 commit, 1 branch, 0 releases, and 1 contributor. The "Code" tab is selected, showing a file named "README.md" with the content "chapter-1-assessment-jmfarrow". A modal dialog is open over the "Clone or download" button, showing the "Clone with HTTPS" option and the URL `https://github.com/CBDRH-HDAT9800/chapter-1-assessment-jmfarrow.git`. The dialog also includes options to "Open in Desktop" and "Download ZIP".

1 Thematic n Leaflet for R... R and GIS ... Formats Introduction... Exercises HDFS Users... Git - Git Refe... CBDRH/Intro... GitHub Class... CBDRH-HDA...

Search or jump to... Pull requests Issues Marketplace Explore

CBDRH-HDAT9800 / chapter-1-assessment-jmfarrow Private Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

chapter-1-assessment-jmfarrow created by GitHub Classroom Edit

Add topics

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

jmfarrow first commit

README.md first commit

README.md

chapter-1-assessment-jmfarrow

Clone with HTTPS Use SSH

Use Git or checkout with SVN using the web URL.

`https://github.com/CBDRH-HDAT9800/chapter-1-assessment-jmfarrow.git`

Open in Desktop Download ZIP

© 2018 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About



# Cloning the assessment repository

---

It is suggested you make a folder for this course

You can keep all the materials for this course organised within that folder however you like however, *each assessment will need its own folder*

*Do not create git managed projects inside other git projects*

- \* That means don't create one week's folder inside another's

Use the HTTPS URL from GitHub to *clone* the project using RStudio

From then on use the RStudio IDE to *commit* and *push* to GitHub



# Chapter 1 Assessment

---

Get the classroom link from the Chapter 1 Assessment page.

Use this link to create a Chapter 1 Assessment repository

Clone the repository

Edit the *questions.txt* file in the repository to add your answers

*Commit* your changes and *push*



# Chapter 1 assessment

---

*(To be submitted using GitHub Classroom. This slide is for reference only.)*

1. What is your previous R and programming experience? [1 mark for answering; we'd like your input to know where you're at]
2. A colleague comes to you and claims 'I always use a bar chart because it's always the best presentation format.' Is your colleague speaking sense? If so why is the statement right? If not how do you counter the argument? [2 marks]
3. When communicating it is very important to 'know your ...'? Finish this sentence. [1 mark]
4. Why is an automated approach to data manipulation better than a manual approach? Why should we write code to manipulate data rather than do it by hand? [3 marks]
5. What does CRAN stand for and what is it? [2 marks]
6. State one advantage to the literate programming approach? [1 mark]