

# Using Git for Science

Seb James

PSY6422

2020/06/06

Find this at: <https://github.com/ABRG-Models/GitTutorial>

# Introduction

- ▶ This session is about a command-line tool called Git.
- ▶ Git is a tool for managing text, so these slides are naturally text heavy!
- ▶ We'll use it with the help of a website built around Git:  
[github.com](https://github.com)
- ▶ I'll give an overview of Git, including its jargon  
(**clone, commit, checkout...**) and why it's such a useful tool, then we'll go through some example tasks together.

# What is Git?

Git is a **Revision Control** or **Version Control** tool.

Revision control has two main features:

# What is Git?

Git is a **Revision Control** or **Version Control** tool.

Revision control has two main features:

1. Revision control allows you to have different versions of a single file without having to explicitly make copies

# What is Git?

Git is a **Revision Control** or **Version Control** tool.

Revision control has two main features:

1. Revision control allows you to have different versions of a single file without having to explicitly make copies
2. Most revision control tools allow several people to work on the same files

## File versions

I bet you have folders that look like this:

## File versions

I bet you have folders that look like this:

- ▶ Project1/myProgram.r

## File versions

I bet you have folders that look like this:

- ▶ Project1/myProgram.r
- ▶ Project1/myProgram\_old.r
- ▶ Project1/myProgram\_1.r
- ▶ Project1/myProgram\_thisOneWorked.r
- ▶ Project1/myProgram\_whoKnowsWhatThisOnels.r

## File versions

I bet you have folders that look like this:

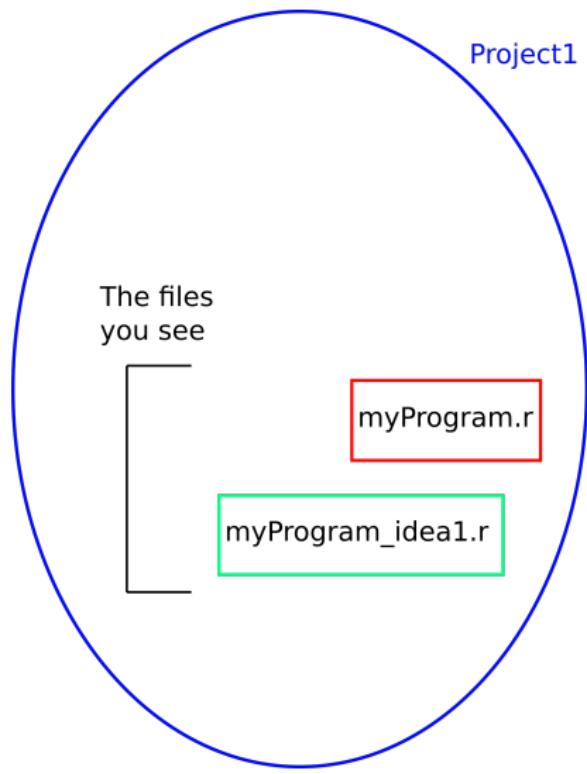
- ▶ Project1/myProgram.r
- ▶ Project1/myProgram\_old.r
- ▶ Project1/myProgram\_1.r
- ▶ Project1/myProgram\_thisOneWorked.r
- ▶ Project1/myProgram\_whoKnowsWhatThisOnes.r

With revision control, you only have

- ▶ Project1/myProgram.r

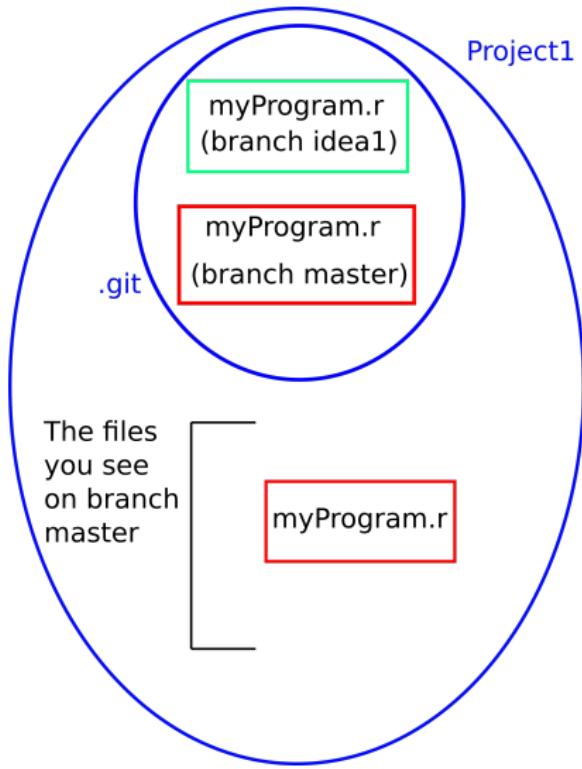
## File versions

- ▶ Ellipses are folders (aka directories)
- ▶ Boxes are files



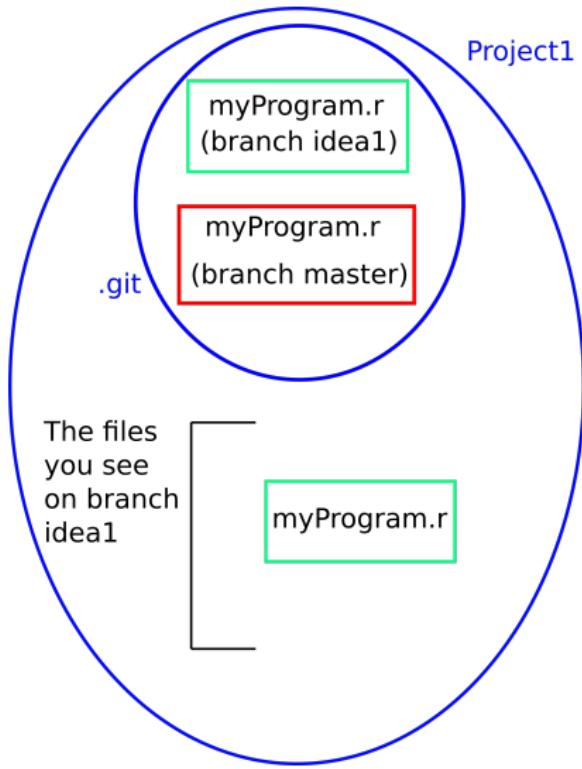
# A file with a view

- ▶ Git presents a view of the file
- ▶ Versions of the file not being viewed are stored away (in `.git`)



# A file with a view

- ▶ Git presents a view of the file
- ▶ Versions of the file not being viewed are stored away (in `.git`)

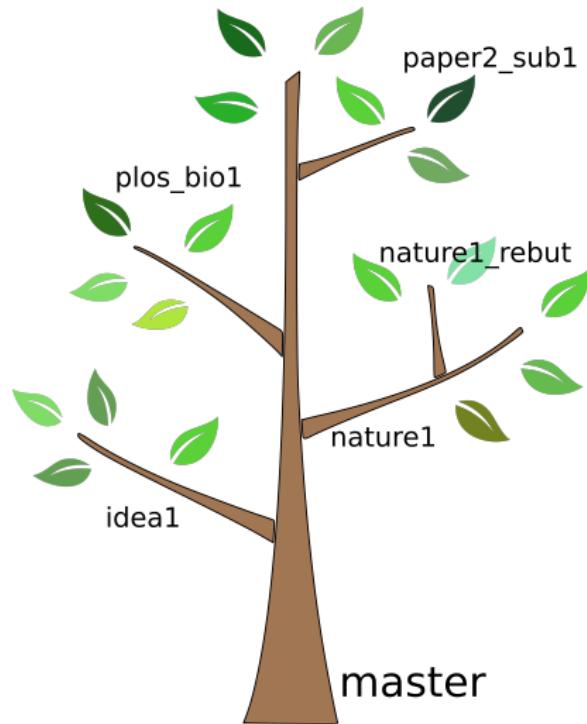


# Branches instead of file versions

When you use git, you use **branches** to work with different file versions. There's one central branch, which is usually called **master**.

## Clone

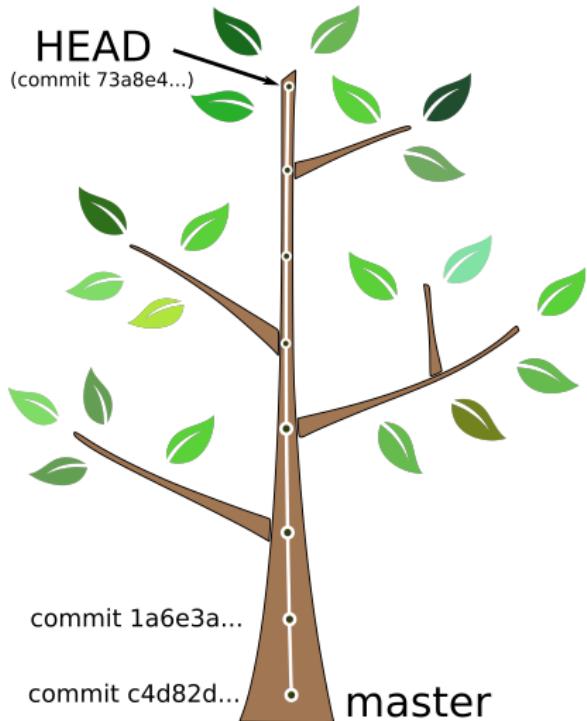
- ▶ When you **clone** a repository from github, you'll get all **myProgram.r** as it exists on the **master** branch
- ▶ Also you get all the information needed to see **myProgram.r** on any of the other branches (each has a name)



# A sequence of changes on master

## Commits

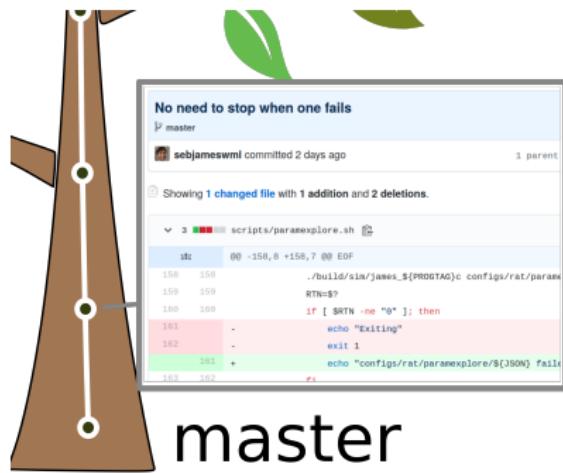
- ▶ There can be different versions of myProgram.r available on **master**; but it's a sequence of changes.
- ▶ Each change is a **commit to master**.
- ▶ Each commit has a universally unique identifier.
- ▶ When you first clone, you'll see the files at the **HEAD** of **master**



# What's in a commit?

Commits contain changes

- ▶ One commit can include the changes to one file
- ▶ One commit can also include changes to multiple files
- ▶ Each commit has a **commit message**

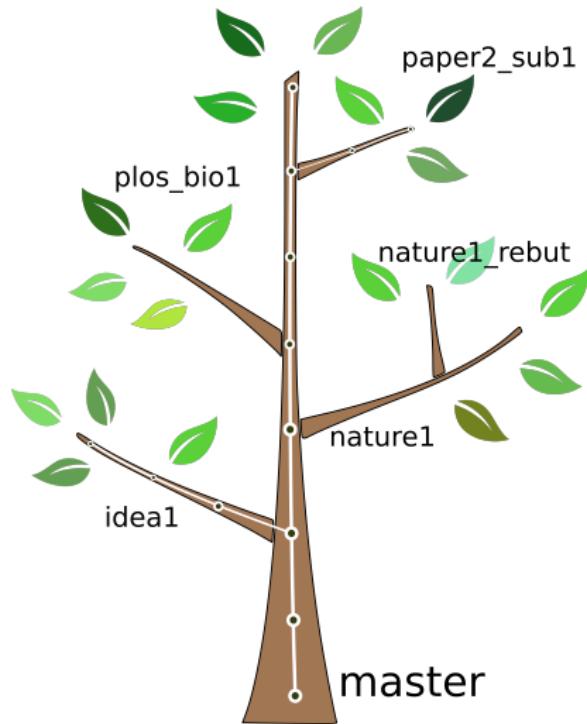


# Checkout a branch

## Checkout

When you **checkout** a branch, it updates your file to the content it has on that particular branch

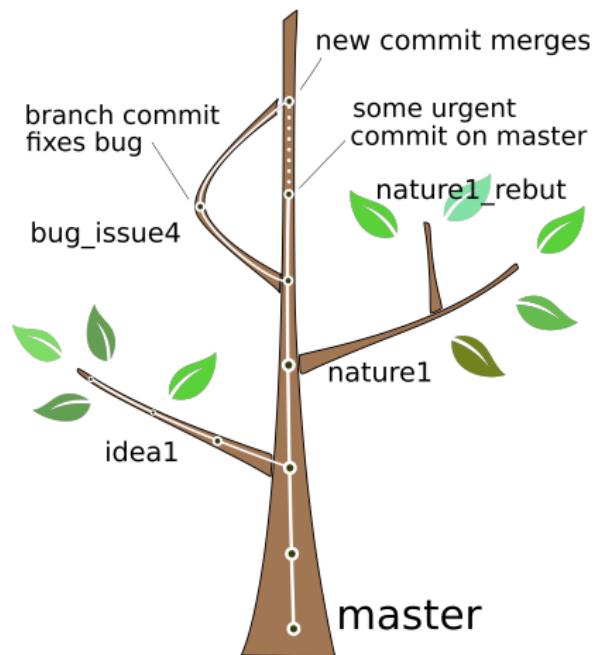
- ▶ Checkout **idea1**; get myProgram.r with the code for your first idea.
- ▶ Checkout **paper2\_sub1**; get myProgram.r as it was when you submitted your second paper based on the project.



# Merge a branch

Here's where the tree analogy begins to break down a little.

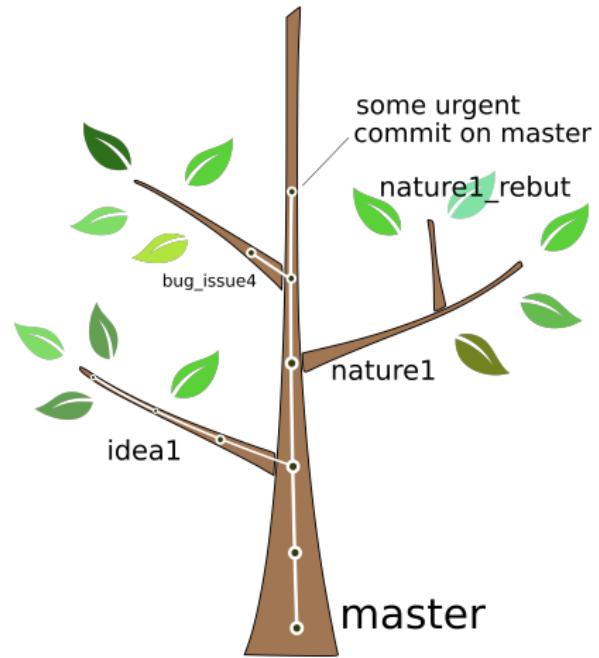
- ▶ Here, a bug was fixed on **bug\_issue4**. While that was happening, someone committed an important commit onto master
- ▶ To bring the bug fix into master, the **bug\_issue4** branch is **merged** into **master** which creates its own commit



# Rebase a branch

Rebase means get out the saw

The developer of the bug in **bug\_issue4** wants to test the bug fix works alongside the urgent commit.

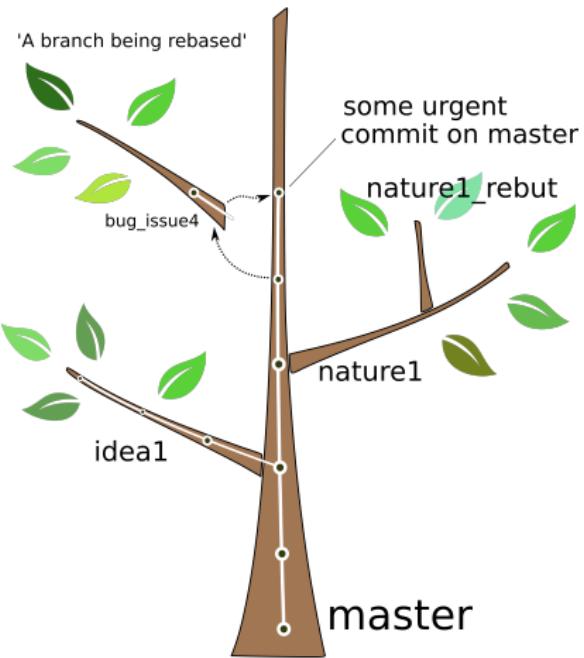


# Rebase a branch

Rebase means get out the saw

The developer of the bug in **bug\_issue4** wants to test the bug fix works alongside the urgent commit.

- ▶ Git has a way to 'saw off' the branch **bug\_issue4** and 'glue it back onto master'

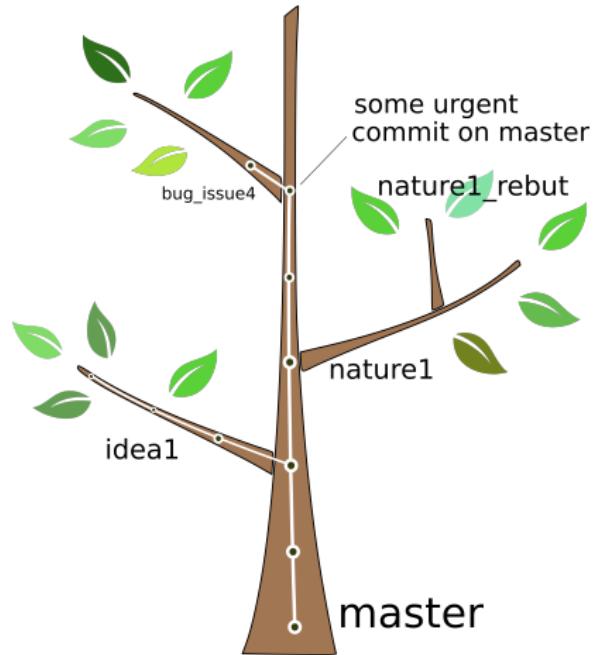


# Rebase a branch

Rebase means get out the saw

The developer of the bug in **bug\_issue4** wants to test the bug fix works alongside the urgent commit.

- ▶ Git has a way to 'saw off' the branch **bug\_issue4** and 'glue it back onto master'
- ▶ Now the bug can be tested, alongside the recent change in **master**, before it is then merged into **master**.
- ▶ Prefer merge over rebase



## Working with other people

This is 'main feature 2'. Now, rewind a little, forget the trees, and think about working with a set of plain files on your computer.

- ▶ Suppose you have some code which is used by yourself and 5 of your colleagues - some sort of library.

## Working with other people

This is 'main feature 2'. Now, rewind a little, forget the trees, and think about working with a set of plain files on your computer.

- ▶ Suppose you have some code which is used by yourself and 5 of your colleagues - some sort of library.
- ▶ You find an error and fix it. Now you have to email the fix to 5 people.

## Working with other people

This is 'main feature 2'. Now, rewind a little, forget the trees, and think about working with a set of plain files on your computer.

- ▶ Suppose you have some code which is used by yourself and 5 of your colleagues - some sort of library.
- ▶ You find an error and fix it. Now you have to email the fix to 5 people.
- ▶ Just after you emailed them, you find an error in your fix, and you have to send another email...

## Working with other people

This is 'main feature 2'. Now, rewind a little, forget the trees, and think about working with a set of plain files on your computer.

- ▶ Suppose you have some code which is used by yourself and 5 of your colleagues - some sort of library.
- ▶ You find an error and fix it. Now you have to email the fix to 5 people.
- ▶ Just after you emailed them, you find an error in your fix, and you have to send another email...
- ▶ Now imagine that one of your colleagues found a separate fix in the same file and emails that around. Which fix is more important? Who is going to merge the two fixes together?

This is where a central copy of the repository becomes important.

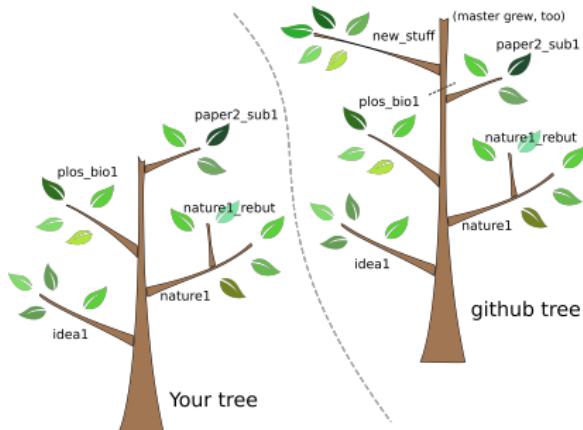
# Fetch and pull

Now imagine there is a shared tree living at `github.com`.

## Fetch

When you **fetch** new changes to the **repo**, information about new branches and commits that were added are copied into your local copy of the repo.

- ▶ Any of the existing branches could have extended
- ▶ New branches could have appeared
- ▶ You can **pull** a branch and that will always do a **fetch** first



# Push your commits

You worked on a new feature, using a new branch. You changed some existing files and added some new ones. You're ready to copy that back to github.com.

## Push

The process of copying new commits and branches as achieved with git **push**.

- ▶ You make sure you have **committed** your changes to your branch
- ▶ You **push** your branch to the online repository (i.e. github)
- ▶ If there are changes on your branch on the online repository that you don't have yet, then you'll have to pull first, **merge** changes and then push.

# Other revision control systems

Git is not the only game in town. Others include:

- ▶ RCS (Revision Control System)
- ▶ SCCS (Source Code Control System)
- ▶ CVS (Concurrent Versions System)
- ▶ Subversion
- ▶ Tons of proprietary systems
- ▶ Bazaar
- ▶ BitKeeper
- ▶ Mercurial

## Other revision control systems

Git is not the only game in town. Others include:

- ▶ RCS (Revision Control System)
- ▶ SCCS (Source Code Control System)
- ▶ CVS (Concurrent Versions System)
- ▶ Subversion
- ▶ Tons of proprietary systems
- ▶ Bazaar
- ▶ BitKeeper
- ▶ Mercurial

Git is not the first revision control system, and its developers could draw on a lot of collective knowledge when designing it.

# Why did someone develop Git?

# Why did someone develop Git?

- ▶ Most revision control tools have been pretty good at feature 1 (file versioning)

## Why did someone develop Git?

- ▶ Most revision control tools have been pretty good at feature 1 (file versioning)
- ▶ ...but not great at managing multiple contributions

## Why did someone develop Git?

- ▶ Most revision control tools have been pretty good at feature 1 (file versioning)
- ▶ ...but not great at managing multiple contributions
- ▶ That caused Linus Torvalds to commit heresy and use the **proprietary** BitKeeper from 2002 to manage the Linux code base.

## Why did someone develop Git?

- ▶ Most revision control tools have been pretty good at feature 1 (file versioning)
- ▶ ...but not great at managing multiple contributions
- ▶ That caused Linus Torvalds to commit heresy and use the **proprietary** BitKeeper from 2002 to manage the Linux code base.
- ▶ In 2005 Linus fell out with BitMover Inc., and Git was created to replace it (and so all was well again in the world of free software OS development - git is fully free).

## Why did someone develop Git?

- ▶ Most revision control tools have been pretty good at feature 1 (file versioning)
- ▶ ...but not great at managing multiple contributions
- ▶ That caused Linus Torvalds to commit heresy and use the **proprietary** BitKeeper from 2002 to manage the Linux code base.
- ▶ In 2005 Linus fell out with BitMover Inc., and Git was created to replace it (and so all was well again in the world of free software OS development - git is fully free).
- ▶ The name git doesn't really mean anything.

# What's different about Git?

- ▶ Git is a distributed revision control system

## What's different about Git?

- ▶ Git is a distributed revision control system
- ▶ It doesn't have the classical client-server architecture...

## What's different about Git?

- ▶ Git is a distributed revision control system
- ▶ It doesn't have the classical client-server architecture...
- ▶ When you **clone** a repository from a source, you have everything (all the file history and meta-data) in those files to become a source for someone else.

# What's different about Git?

- ▶ Git is a distributed revision control system
- ▶ It doesn't have the classical client-server architecture...
- ▶ When you **clone** a repository from a source, you have everything (all the file history and meta-data) in those files to become a source for someone else.
- ▶ That means you can work on your code, making incremental **commits** even when you don't have internet access.
- ▶ And every copy of the **repo** is a backup!

# What's different about Git?

- ▶ Git is a distributed revision control system
- ▶ It doesn't have the classical client-server architecture...
- ▶ When you **clone** a repository from a source, you have everything (all the file history and meta-data) in those files to become a source for someone else.
- ▶ That means you can work on your code, making incremental **commits** even when you don't have internet access.
- ▶ And every copy of the **repo** is a backup!
- ▶ It also means you can '**fork**' a repository

# What's different about Git?

- ▶ Git is a distributed revision control system
- ▶ It doesn't have the classical client-server architecture...
- ▶ When you **clone** a repository from a source, you have everything (all the file history and meta-data) in those files to become a source for someone else.
- ▶ That means you can work on your code, making incremental **commits** even when you don't have internet access.
- ▶ And every copy of the **repo** is a backup!
- ▶ It also means you can '**fork**' a repository
- ▶ Typically you will work with a common **remote** repository (github) as your **upstream** source.

## Git is not github.com

- ▶ Github is a commercial website which makes it easy to use Git
- ▶ bitbucket.org is an alternative

## Git is not github.com

- ▶ Github is a commercial website which makes it easy to use Git
- ▶ bitbucket.org is an alternative
- ▶ Generally, public hosting is free, ~~private hosting incurs a fee~~

## Git is not github.com

- ▶ Github is a commercial website which makes it easy to use Git
- ▶ bitbucket.org is an alternative
- ▶ Generally, public hosting is free, ~~private hosting incurs a fee~~
- ▶ It's pretty easy to host a git repository yourself, but the nice web interface has made github.com very popular for source code hosting

## Git is not github.com

- ▶ Github is a commercial website which makes it easy to use Git
- ▶ bitbucket.org is an alternative
- ▶ Generally, public hosting is free, ~~private hosting incurs a fee~~
- ▶ It's pretty easy to host a git repository yourself, but the nice web interface has made github.com very popular for source code hosting
- ▶ It's now a big business; it was acquired by Microsoft in 2018

## Why is Git good for us?

We don't have hundreds of people working on the same files, and often our work is carried out individually, but...

## Why is Git good for us?

We don't have hundreds of people working on the same files, and often our work is carried out individually, but...

- ▶ We write code, so that's very natural to hold in revision control

## Why is Git good for us?

We don't have hundreds of people working on the same files, and often our work is carried out individually, but...

- ▶ We write code, so that's very natural to hold in revision control
- ▶ We store experimental data which is valuable and needs to be protected

## Why is Git good for us?

We don't have hundreds of people working on the same files, and often our work is carried out individually, but...

- ▶ We write code, so that's very natural to hold in revision control
- ▶ We store experimental data which is valuable and needs to be protected
- ▶ We write papers about the data and the code

## Why is Git good for us?

We don't have hundreds of people working on the same files, and often our work is carried out individually, but...

- ▶ We write code, so that's very natural to hold in revision control
- ▶ We store experimental data which is valuable and needs to be protected
- ▶ We write papers about the data and the code
- ▶ We have a frequent need to **tag** our work (e.g. to match up with a paper or document)

## Why is Git good for us?

We don't have hundreds of people working on the same files, and often our work is carried out individually, but...

- ▶ We write code, so that's very natural to hold in revision control
- ▶ We store experimental data which is valuable and needs to be protected
- ▶ We write papers about the data and the code
- ▶ We have a frequent need to **tag** our work (e.g. to match up with a paper or document)
- ▶ “It used to work, but now I've broken it and I can't get it back to working again”: Revision control makes it easy to revert to a version of your code which you know will work

## Why is Git good for us?

We don't have hundreds of people working on the same files, and often our work is carried out individually, but...

- ▶ We write code, so that's very natural to hold in revision control
- ▶ We store experimental data which is valuable and needs to be protected
- ▶ We write papers about the data and the code
- ▶ We have a frequent need to **tag** our work (e.g. to match up with a paper or document)
- ▶ "It used to work, but now I've broken it and I can't get it back to working again": Revision control makes it easy to revert to a version of your code which you know will work
- ▶ You can include your paper (and your data) alongside your model code in a single, public repository
- ▶ Use of Github is a very effective way to share your published models with your peers

# The rest of the session

- ▶ Create repository on github, clone, add something
- ▶ git add, git move, git remove, git branch -D etc
- ▶ git checkout -b newbranch
- ▶ Github READMEs and markdown
- ▶ Github issues
- ▶ doxygen and codedocs.xyz
- ▶ Demonstration of branches for papers (BarrelEmerge)

For a good hands-on tutorial, head over to:

<http://sebjameswml.github.io/git-novice/>

or try the more up to date version of that 'Software Carpentry' resource at:

<http://swcarpentry.github.io/git-novice/>