Hydro-JULES Education

FDRI FLOODS & DROUGHTS RESEARCH INFRASTRUCTURE

UK Centre for Ecology & Hydrology

NATIONAL CAPABILITY FOR UK CHALLENGES

NATIONAL CAPABILITY FOR GLOBAL CHALLENGES

CANARI

CCCENTRE Climate Biodiversity Water

Intro to Git

Simon Stanley

UK Centre for Ecology & Hydrology

Contents

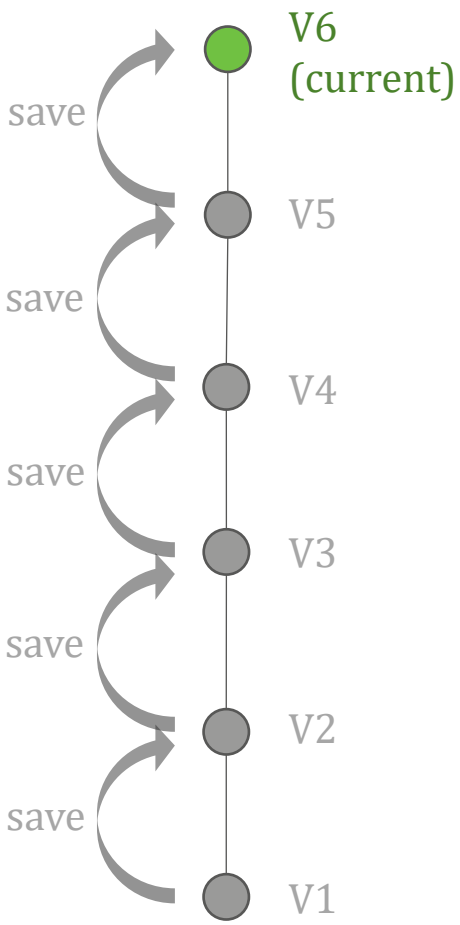
- Version control basics
- Branching and merging
- Remote repositories
- Further tools

Version control

Every time we update and save a file, we create a new version of that file.

In the traditional way we do this, only the **latest version** is available.

- We have no access to previous versions.
- We cannot see the changes made in each save.

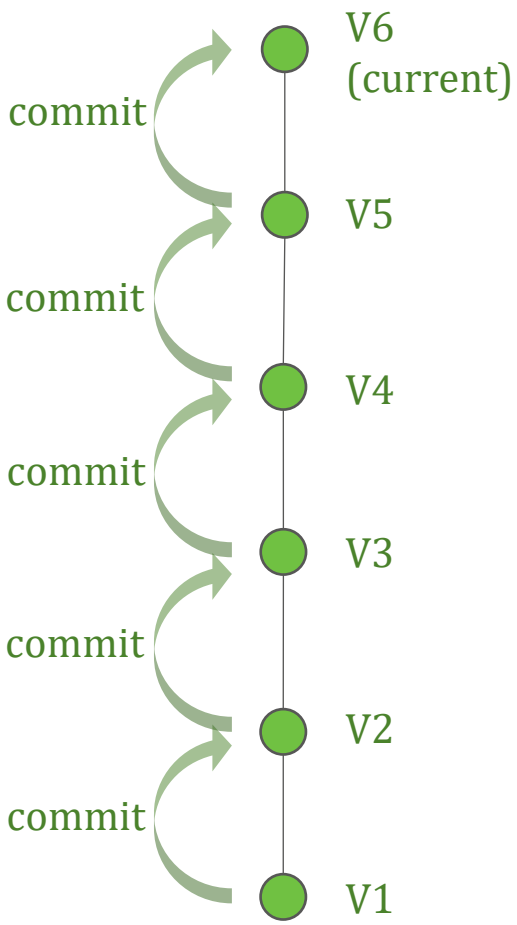


Version control

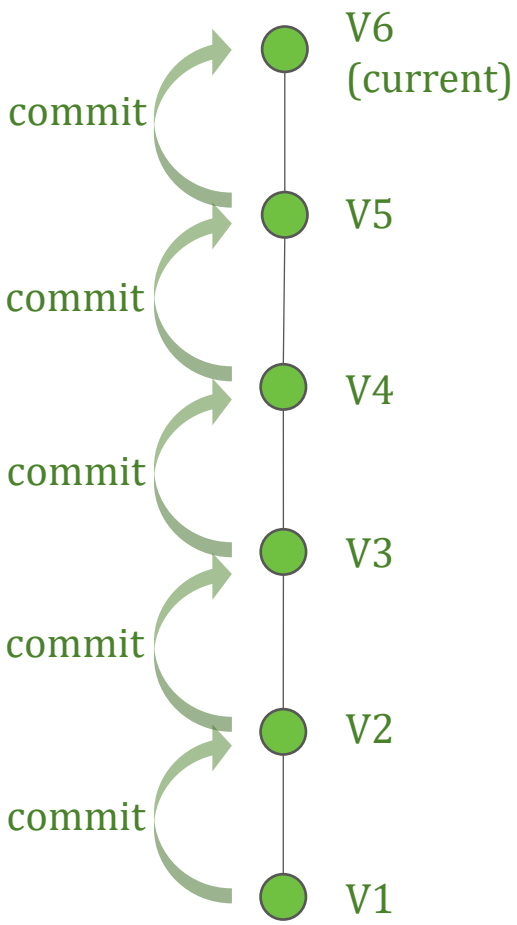
Version control software (like Git) gives us these capabilities to see and interact with every change…

… and much more!

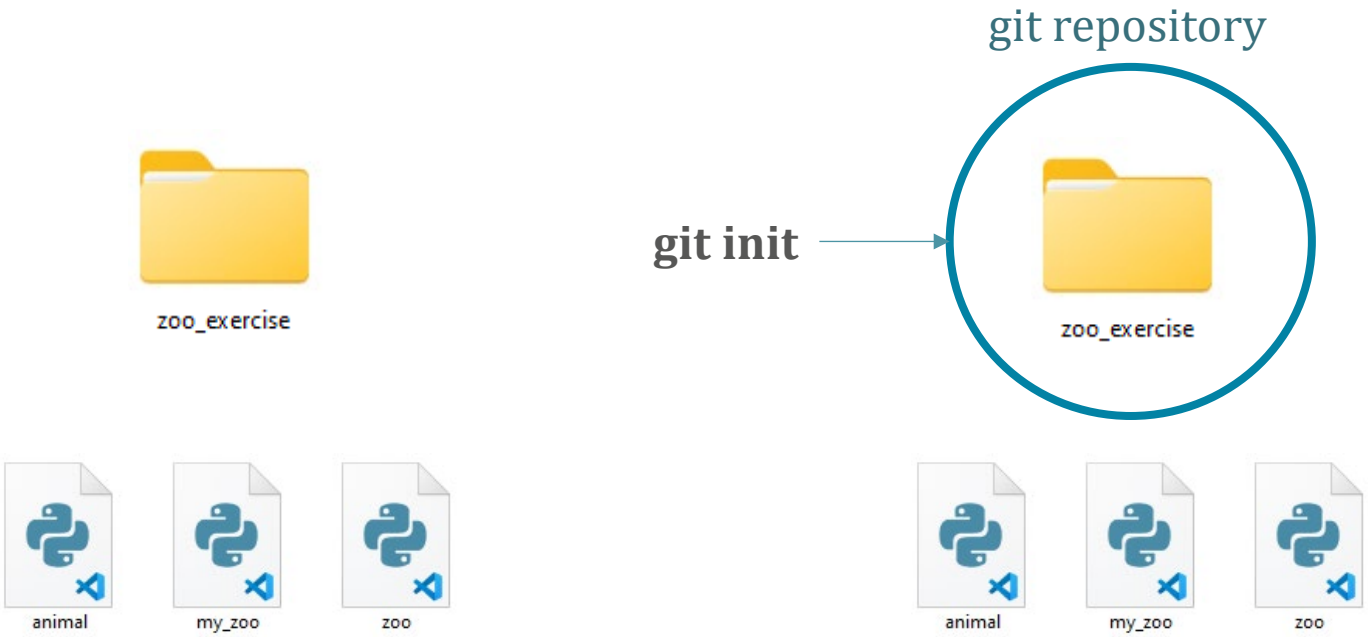
Let's look at a demonstration to see why this is useful.



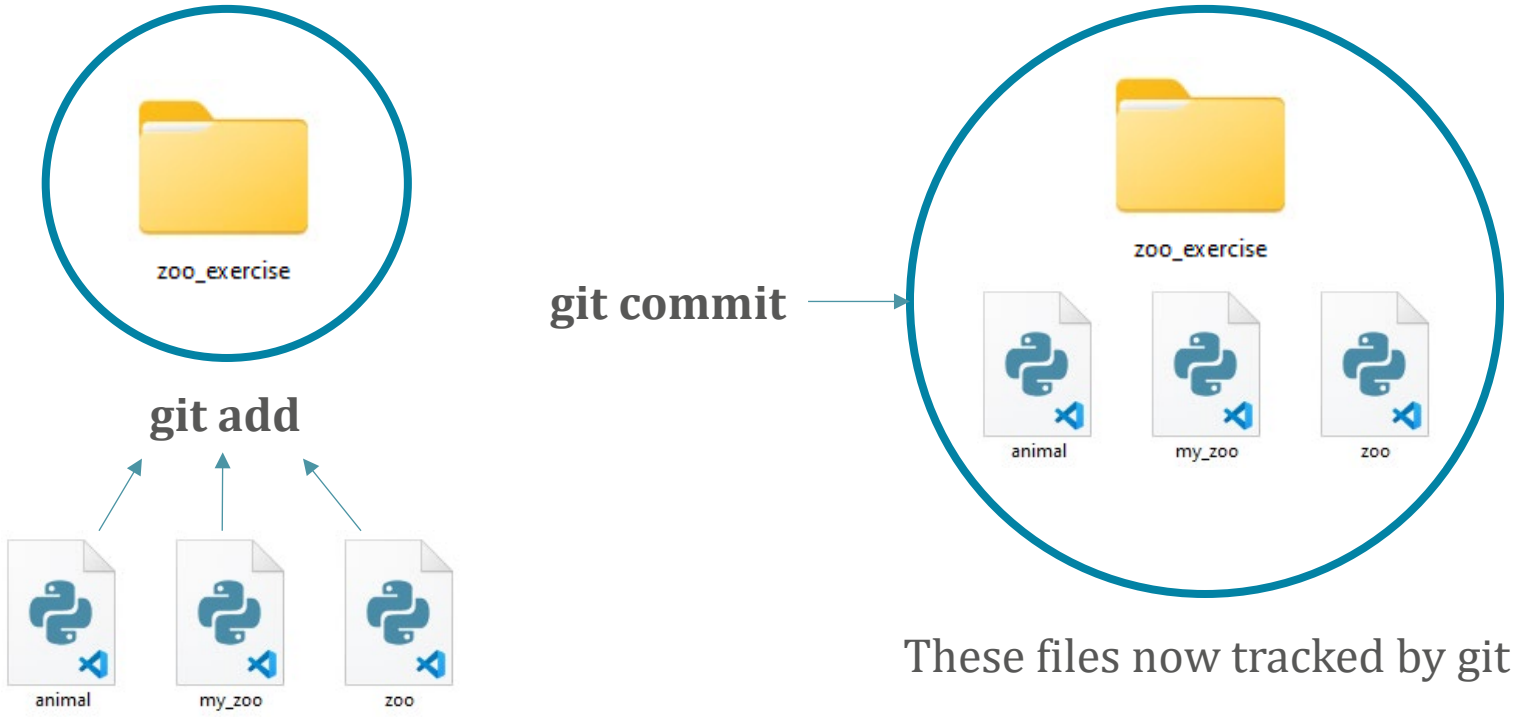
Demo



Demo – Initialise repository



Demo – Initialise repository



These files now tracked by git

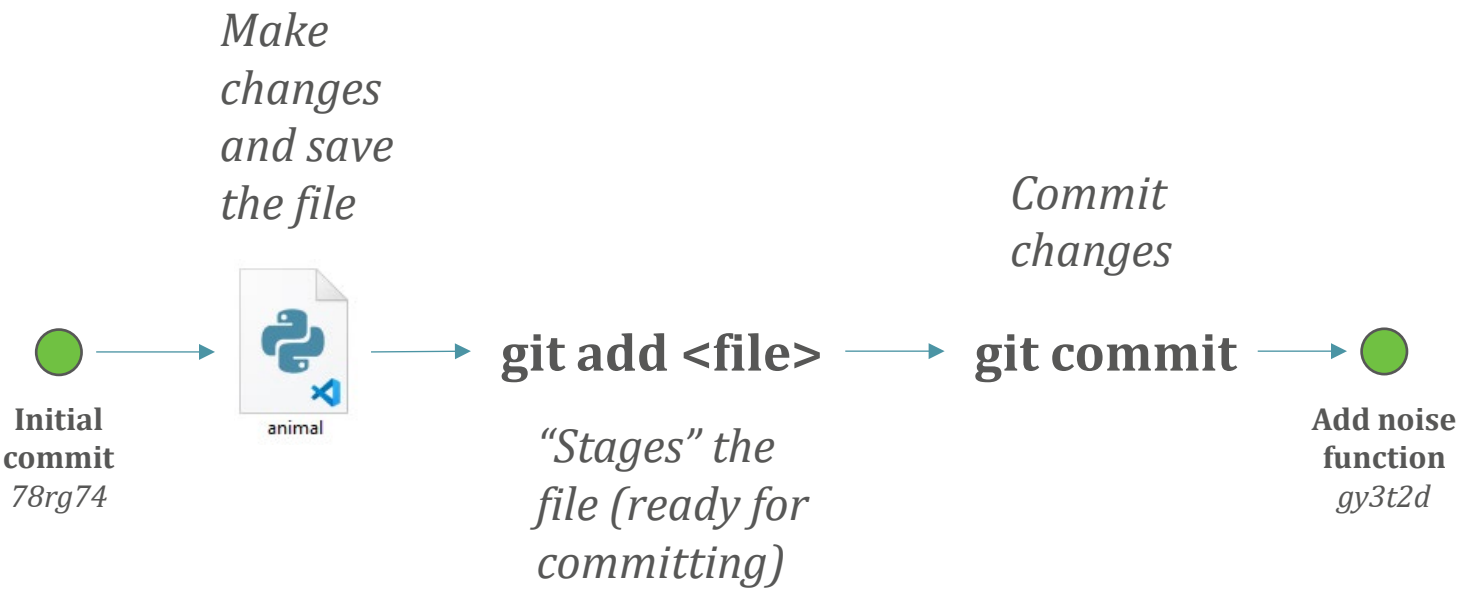
Demo – Explore changes

Make changes and save the file

git status — *Shows the status of the repository, including any files that have been changed since last commit*

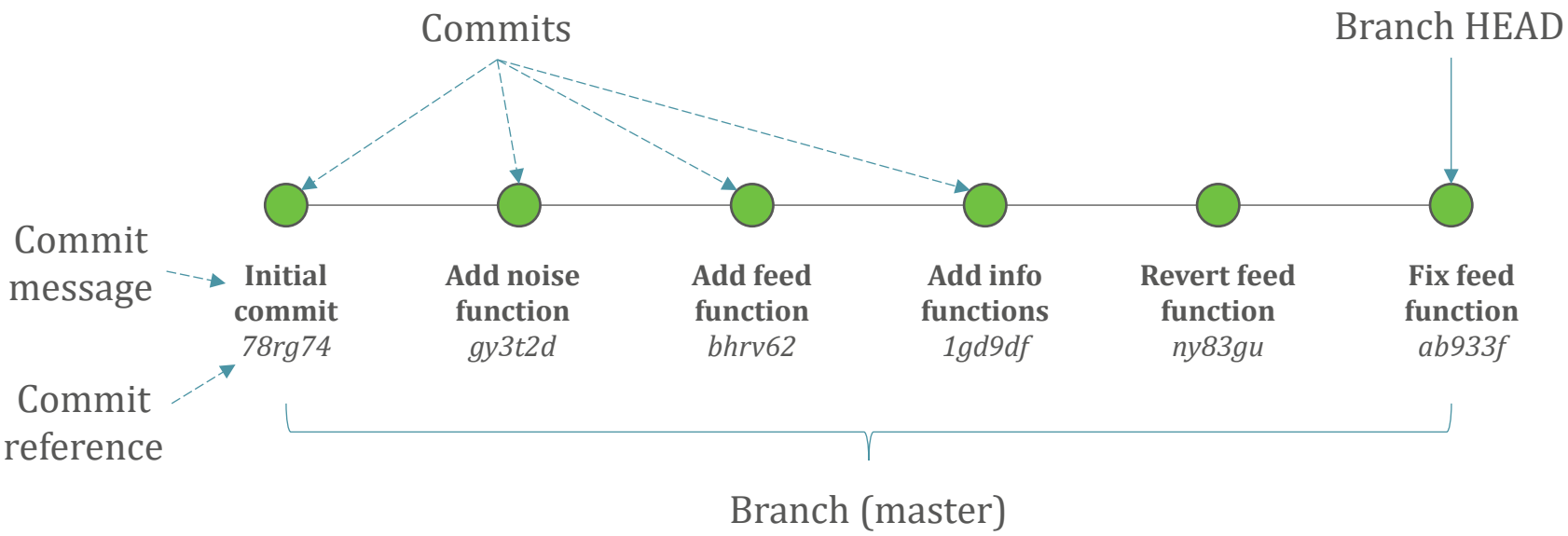
git diff — *Shows the actual changes that have been made since last commit*

Demo – Commit changes



Tip: use **git commit -m "<Commit message>"**

Demo - Branch



git log *to see all commits*

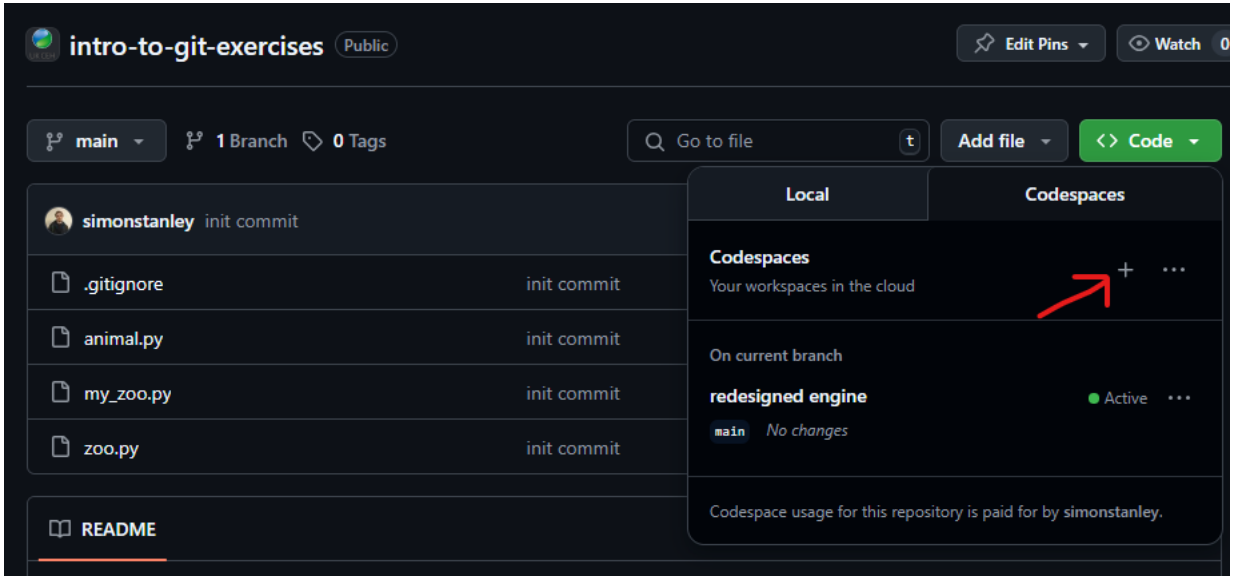
git show *to inspect individual commit*

Exercise 1

Log into GitHub:

https://github.com/NERC-CEH/intro-to-git-exercises

Click “Code” -> “Codespaces” -> “+”



Exercise 1

1. Add noise function
   - Use **git status** and **git diff** to check the changes.
   - Use **git add** and **git commit** to commit the changes.
2. Add feed function
3. Add info function and list_animals function
   - Add and commit both files into one commit
4. Checkout one of the earlier commits
5. Checkout the latest commit
6. Revert one of the commits (not the initial commit!)

Commands

• **git init** – Set a directory as a git repository	**Initialisation**
• **git status** – See overall state of repository • **git log** – List all commits • **git show** – Show changes in a commit	**Information**
• **git checkout <ref>** – Move between commits	**Movement**
• **git diff** – Show uncommitted changes • **git add <file>** – Stage uncommitted changes • **git commit** – Commit added changes • **git revert** – Undo changes in a commit	**Committing changes**

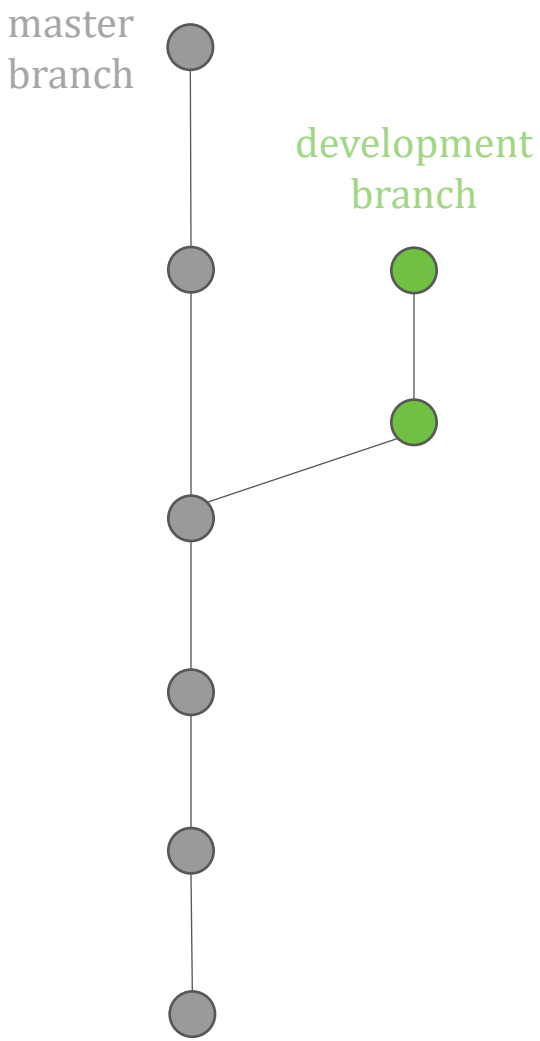
Branching

So far, we have made all out commits on one branch, the master branch.

In git we can create new branches which contain their own chain of commits.

Branching allows us to develop code in isolation, meaning we cannot accidently break things on the master branch.

It means we can test and experiment to our hearts content!



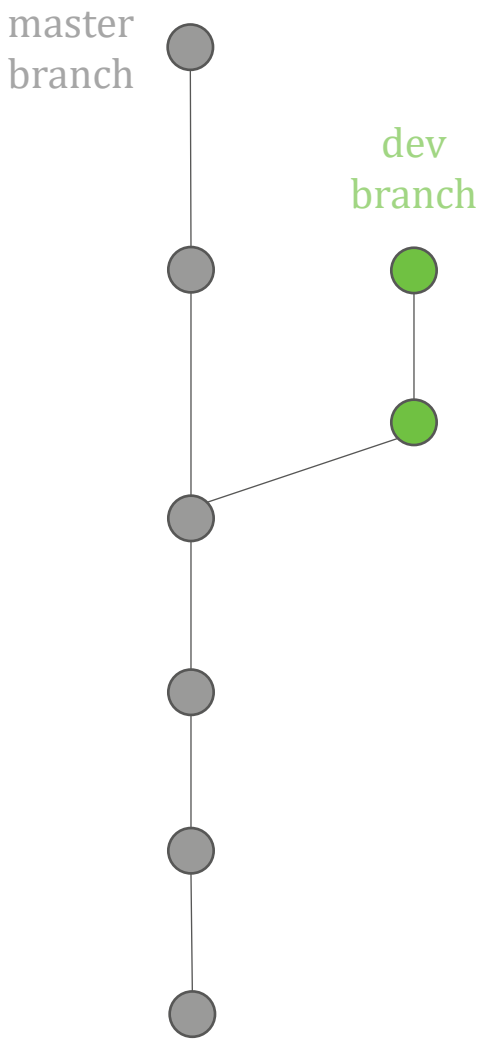
Branching

There are three key commands for branching:

git branch

Lists all the branches in the repository. In this example it would list:

- master
- dev

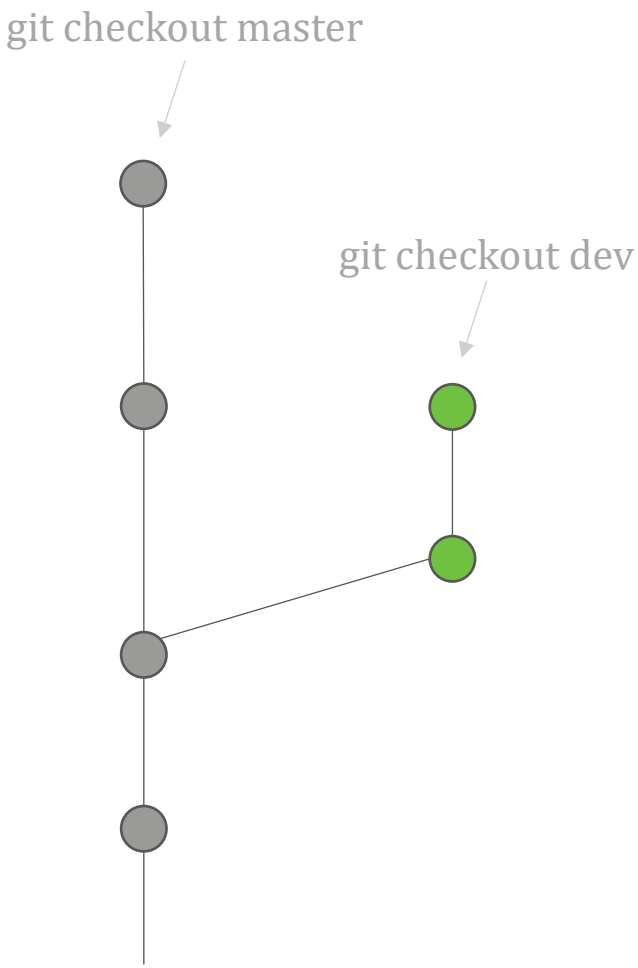


Branching

git checkout <branch name>

Switches us onto the chosen branch.

Remember, when we are “on a branch”, it means the directory is changed to the state of the latest commit on that branch.



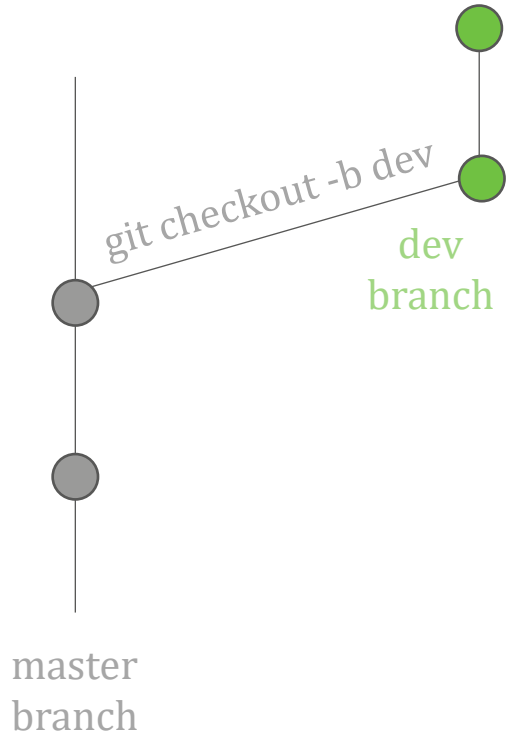
Branching

git checkout -b <branch name>

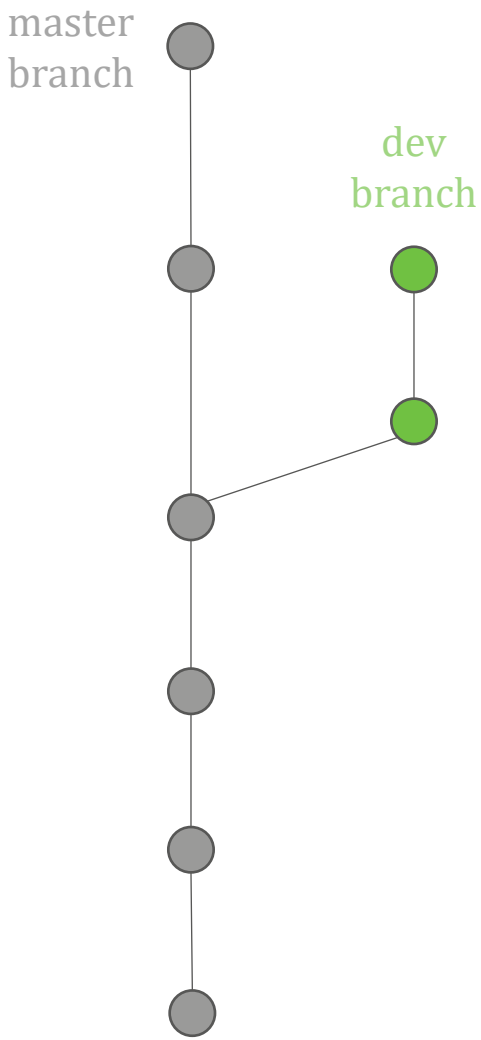
This creates a new branch from the current commit.

Note, the new branch starts in the same state as where it was branched from.

When you start making commits on that branch, it will grow independently from the original.

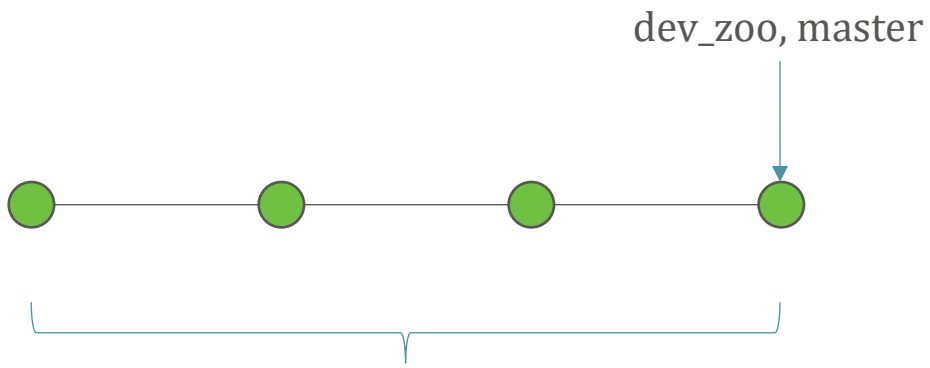


Demo



Demo – New branch

git checkout -b dev_zoo

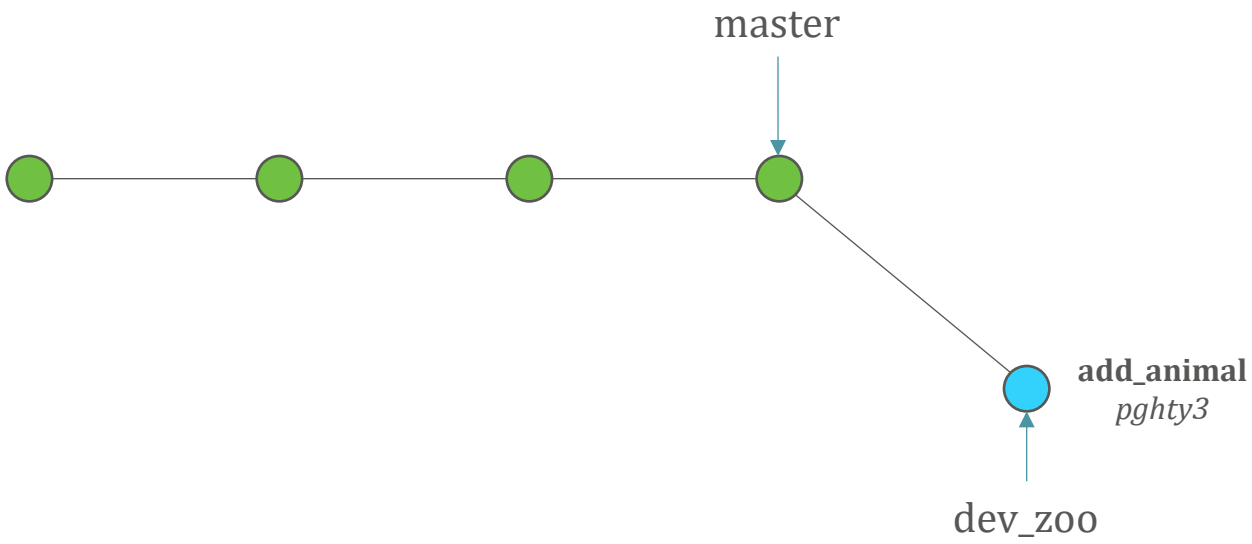


These commits are now part of master **and** dev_zoo.

Demo – New commit

Added new commit on dev_zoo

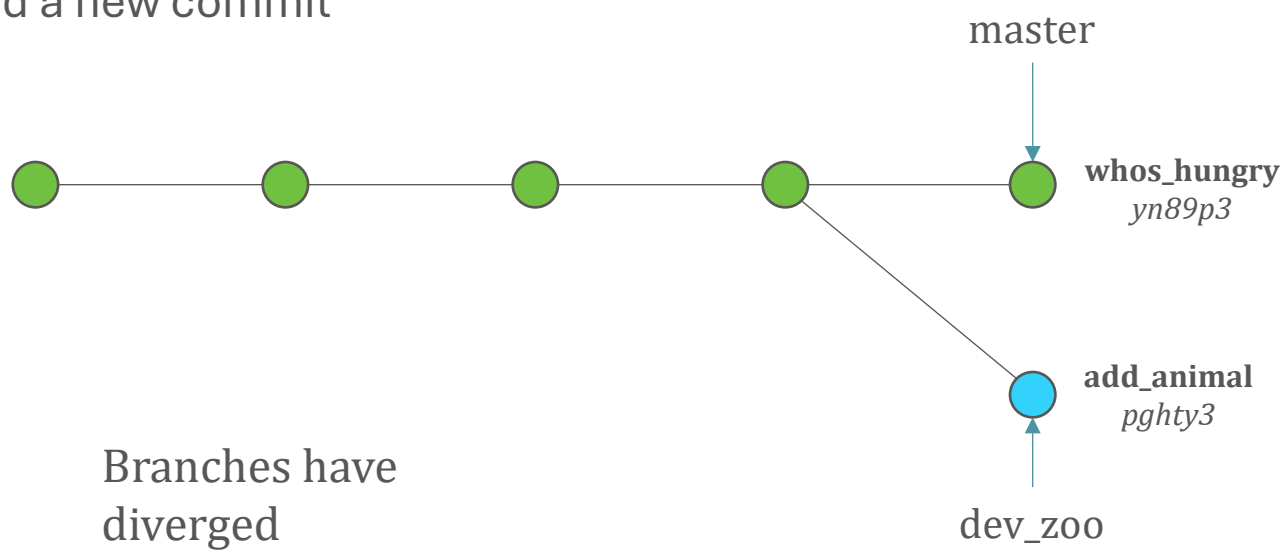
Master branch unchanged



Demo – Commit on master

git checkout master

Added a new commit



Branches have diverged

Merging

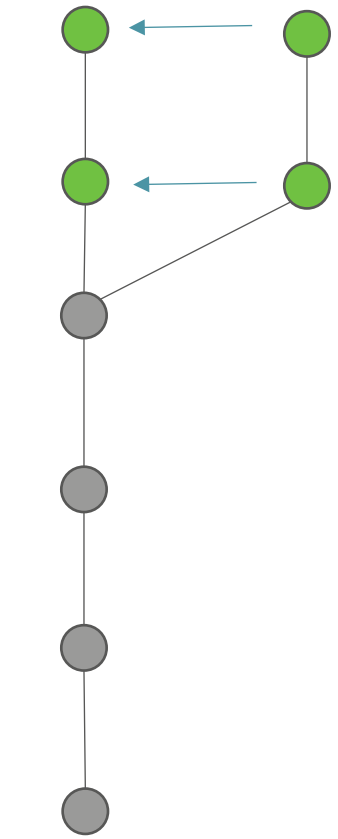
Branching allows us to develop code in isolation and safety.

Once we are happy our developments are correct, we want to merge these back onto the master branch.

This is done with the command:

git merge <branch to merge>

git merge <branch>



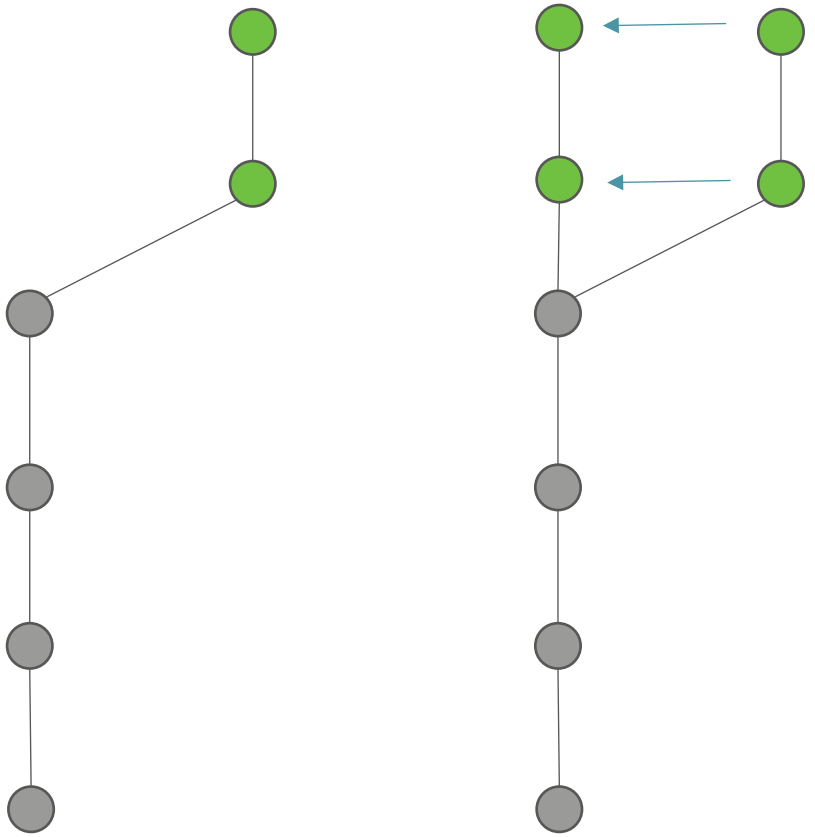
Merging

There are two types of merge:

1. **Fast forward merging**

Where no change has been made to the original branch, the commits can just be added on… (same as if they were developed on that branch in the first place)

git merge <branch>



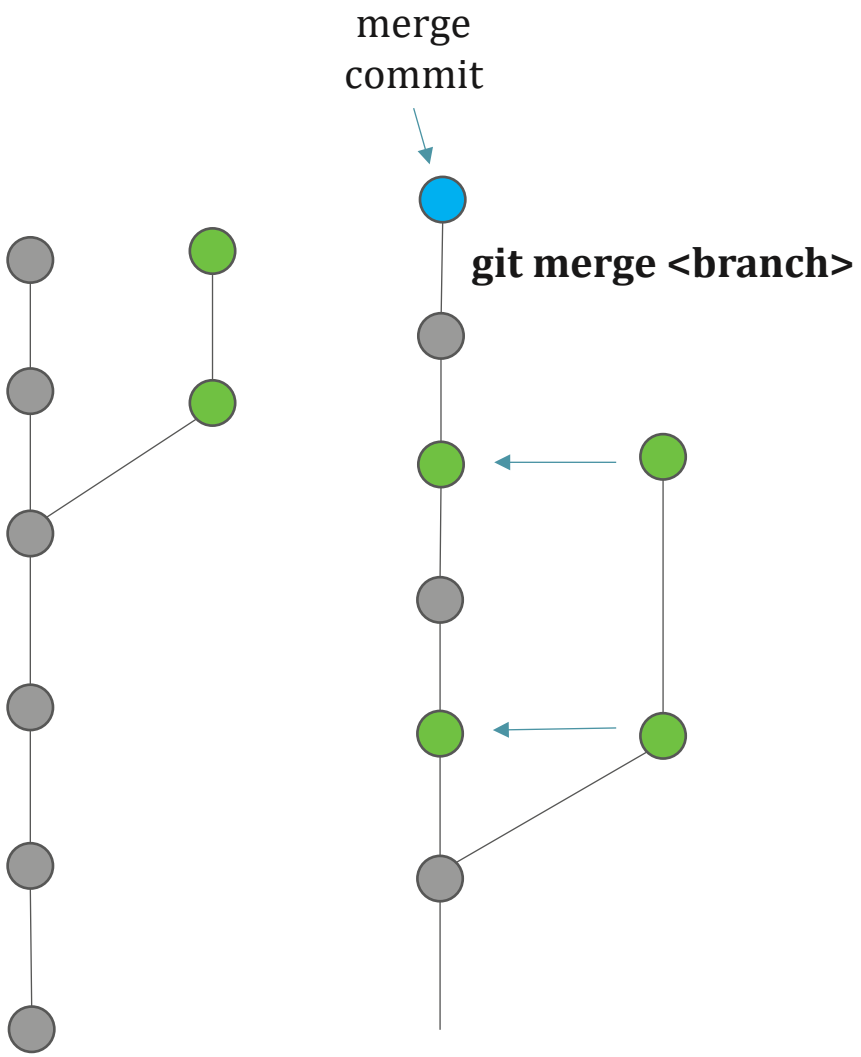
Merging

The second type of merge:

2. Auto merging

Where other commits have been made (or merged into) the original branch, git must work out how to combine the changes.

In doing so it creates an extra merge commit.



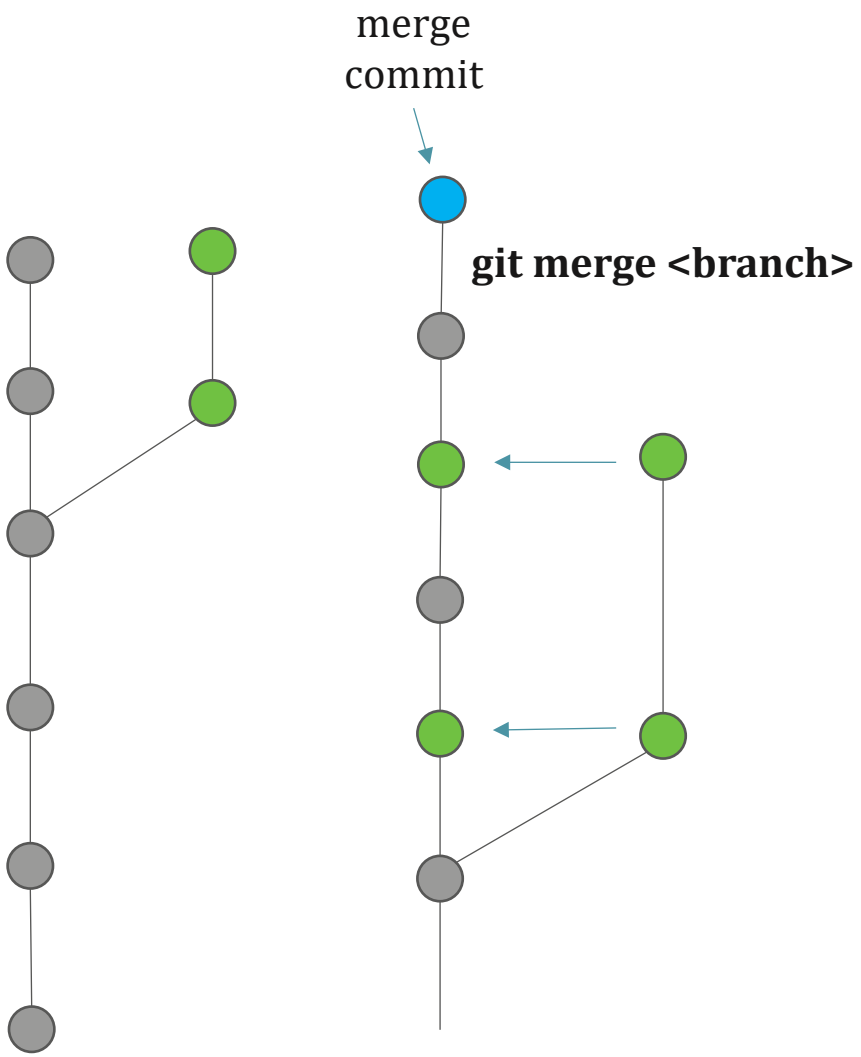
Merging

The second type of merge:

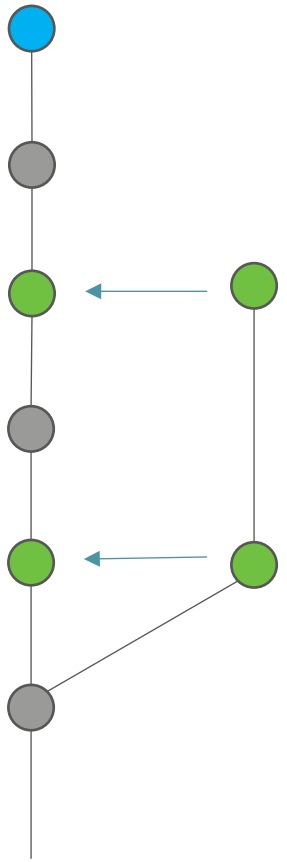
2. Auto merging

In most circumstances you don't need to worry about this, git does the work.

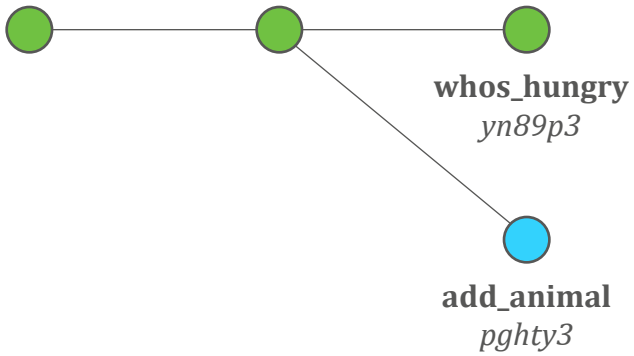
However, when separate developments change the same part of the file, we get a **conflict**…



Demo



Demo – Merging



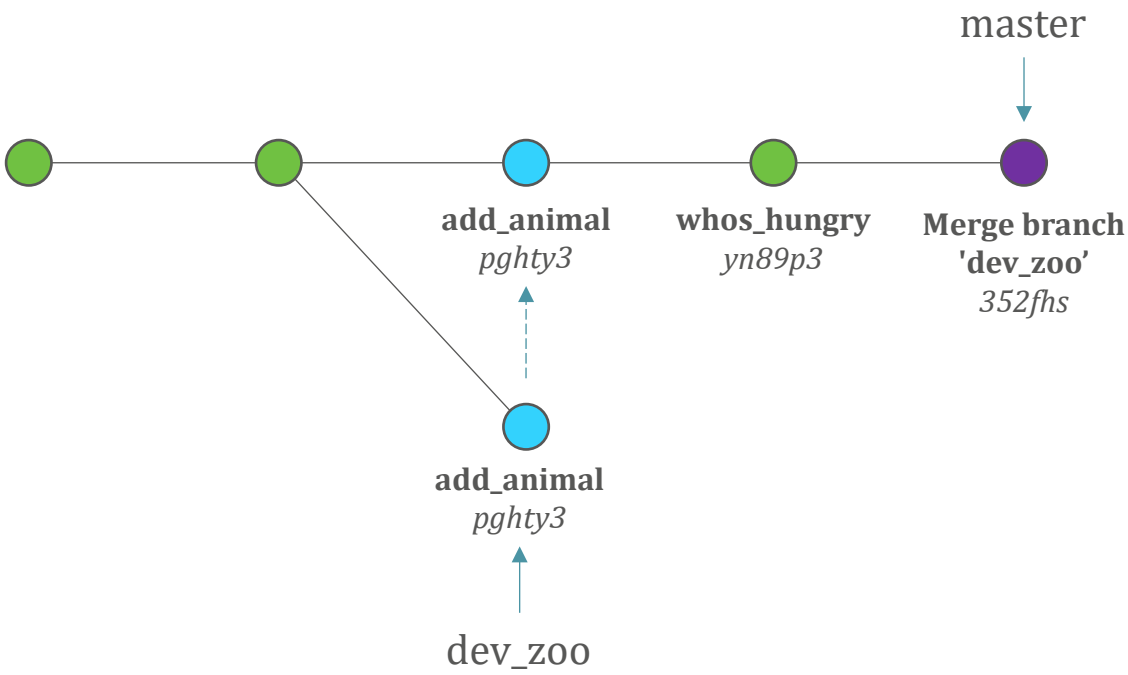
To merge dev_zoo into master…

git checkout master

git merge dev_zoo

Demo – Merging

git merge dev_zoo



Exercise 2

1. Create a new branch
   - Use **git checkout –b <branch>**
2. Add (and commit) the add_animal function
3. Switch between master and new branch
   - Use **git branch** to see all the branches
   - Use **git checkout <branch>** to switch
4. Merge new branch into master
   - Switch to master branch using
   - Use **git merge <branch>** to merge in commits from new branch

Commands

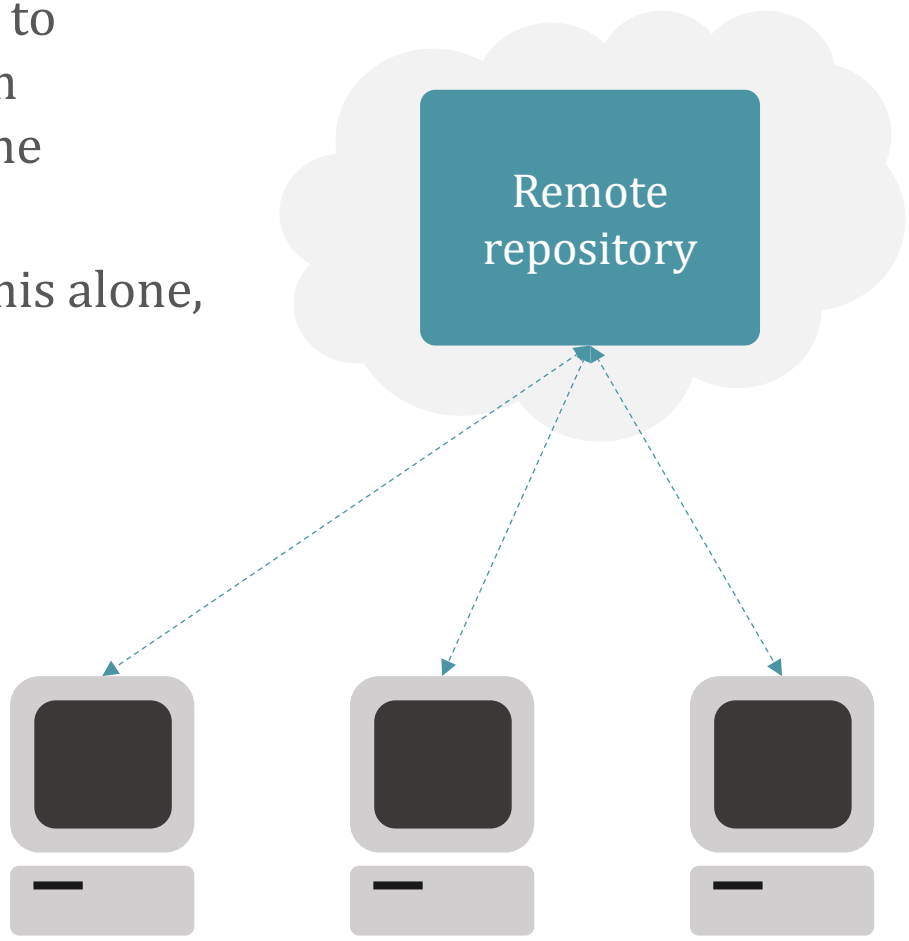
- **git branch** – List all branches
- **git checkout <branch>** – Move between branches
- **git checkout –b <branch>** – Create new branch
- **git merge <branch>** - Merge commits from other branch into current branch

Remote repositories

We have seen how git allows us to develop files in isolated ways on separate branches and merge the changes together.

However, we have been doing this alone, on our own personal machines.

Development jumps to the next level when we can share copies of repositories have multiple people work together in parallel.

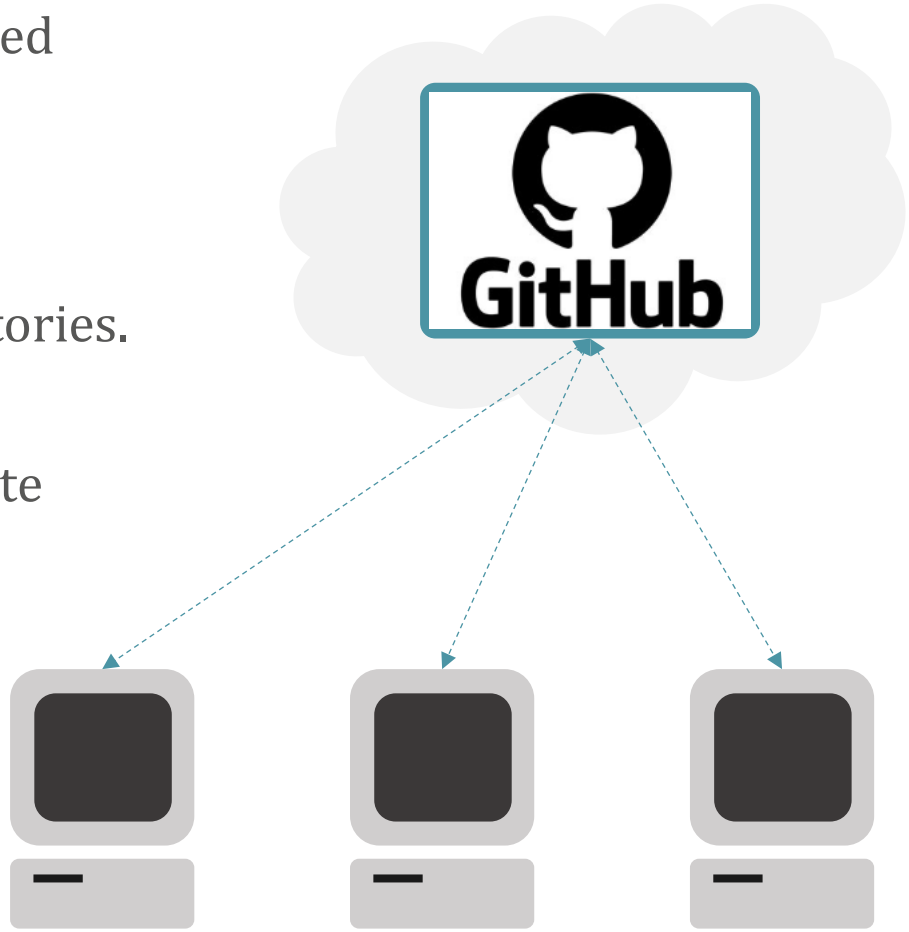


Remote repositories

To do this, we create a centralised version of the repository that everyone can access.

These are called remote repositories.

The most famous host for remote repositories is **GitHub**.

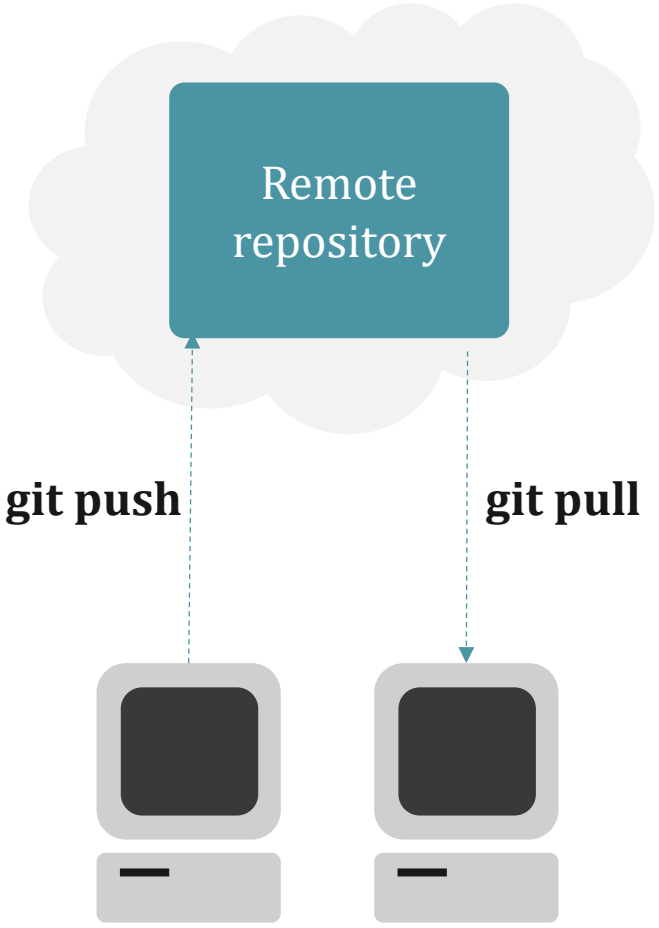


Remote repositories

We won't go into detail in this course, but the main commands used to interact with remote repositories are:

- **git clone** – Download a repository onto your machine
- **git push –** push the latest branches / commits to the remote
- **git pull –** pull down the latest branches / commits (i.e. changes others have made)

These commands allow everyone to stay up to date with the latest developments.



Further tools

Manipulating commits:

git add –patch <files>

This triggers an interactive session where you can pick individual changes within a file to be staged.

git commit –amend

Edit the latest commit, either add more changes to it, or update the commit message.

*Note, this should **NOT** be done after a commit has been pushed to a remote repository.*

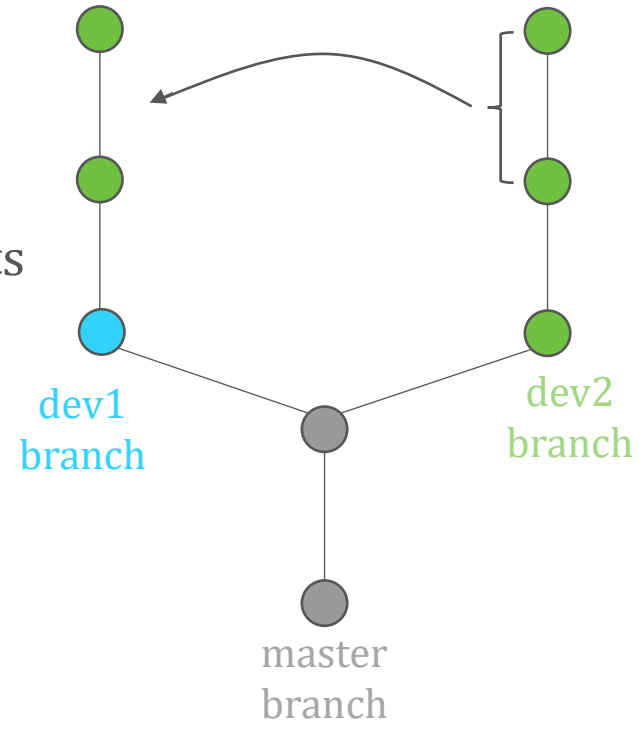
Further tools

Manipulating commits:

git rebase

This is a powerful tool for reorganising commits and branches. The two main capabilities are:

- Squash multiple commits into a single commit.
- Moving sets of commits from one branch to another (like merging, but not the whole branch).



*This should **NOT** be done after a commit has been pushed to a remote repository.*

Further tools

git stash

If you are working on changes, but before finishing and committing, you need to switch branches, e.g. a major bug on the master branch needs fixing ASAP!

Well you can't! git does not allow you to switch branches before changes are committed because they would be lost.

git stash allows you to "stash" uncommitted changes so you can do this. When you run **git stash**, the repository will return to the current state of the branch, and your changes will disappear. You can then switch.

When you are ready to get those uncommitted changes back, use:

git stash apply

Further tools

Investigation:

git blame <file>

This prints out every line of a file with the commit that last changed it. This is useful in debugging for tracking down why a change was made (hence why clear commit messages are important)

git diff <branch1>..<branch2>

This shows the line by line differences between two branches.

git log <branch1>..<branch2>

Shows all the commits that are in branch2 that are not in branch1.

Further tools

Investigation:

git log -S “string”

Finds commits which contain the string either in the commit message or the commit changes.

git grep “string”

Search the repository for a string.

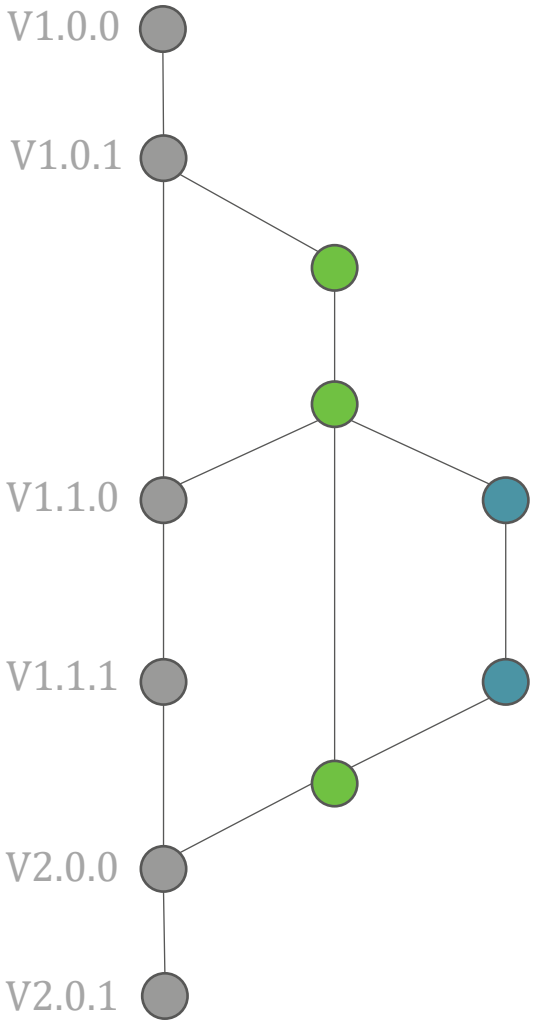
(This is just like the normal grep command but more efficient when grepping inside a repository)

Thank you

Any questions?

Further information

https://git-scm.com/



Installing git

See here for installing git:

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

git init

Any folder can be turned into a git repository.

It simply means any changes to the contents of that folder can now be tracked by git.

To make a folder a git repository:

```
cd /path/to/directory/
git init
```

Example:

```
simsta@WLL-HGVWHM2 MINGW64 ~
$ cd /n/work/NetZero/git_example

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example
$ git init
Initialized empty Git repository in //nercwlctdb/pcusers/PCUSERS1/SIMSTA/work/NetZero/git_example/.git/
```

git status

This command shows us the current state of the git repository. The key information being:

- What branch we are on
- What files have been changed, staged or are untracked

We see what each of these mean as we go.

git status

Example:

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        my_zoo.py
        zoo.py

nothing added to commit but untracked files present (use "git add" to track)
```

git diff

After changes are made, but before they are staged and committed, we use **git diff** to view the updates.

This shows us everything that has been added and removed allowing us to review the changes.

Example

git status

Lists the files that have changed

git diff

Prints out all the changes.

Green for lines added

Red for lines removed

Where a line is changed we see the previous version removed and new version added

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   my_zoo.py
        modified:   zoo.py

no changes added to commit (use "git add" and/or "git commit -a")

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git diff
diff --git a/my_zoo.py b/my_zoo.py
index ddd43e8..b138e5d 100644
--- a/my_zoo.py
+++ b/my_zoo.py
@@ -8,8 +8,11 @@ from zoo import Animal, Zoo
 best_zoo = Zoo()

 # Create some animals and add to the Zoo
-best_zoo.add_animal(Animal(species="lion", name="Fred", legs=4))
-best_zoo.add_animal(Animal(species="lion", name="Kate", legs=4))
-best_zoo.add_animal(Animal(species="bear", name="Pete", legs=2))
-best_zoo.add_animal(Animal(species="bear", name="Suzy", legs=2))
-best_zoo.add_animal(Animal(species="snake", name="John", legs=0))
+best_zoo.add_animal(Animal(species="lion", name="Fred", legs=4, noise="Roooaaaar"))
+best_zoo.add_animal(Animal(species="lion", name="Kate", legs=4, noise="Rooaar"))
+best_zoo.add_animal(Animal(species="bear", name="Pete", legs=2, noise="Grrrrr"))
+best_zoo.add_animal(Animal(species="bear", name="Suzy", legs=2, noise="Grrrraaar"))
+best_zoo.add_animal(Animal(species="snake", name="John", legs=0, noise="Hisssss"))
+
+for animal in best_zoo.animals:
+    animal.make_noise()
diff --git a/zoo.py b/zoo.py
index 2d826b3..2048271 100644
--- a/zoo.py
+++ b/zoo.py
@@ -3,10 +3,15 @@ Code to define animals and zoos

 """
 class Animal(object):
-    def __init__(self, species, name, legs):
+    def __init__(self, species, name, legs, noise):
         self.species = species
         self.name = name
         self.legs = legs
+        self.noise = noise
+
+    def make_noise(self):
+        print(self.noise)
+        print()


 class Zoo(object):
```

UK Centre for Ecology & Hydrology

git commit

Committing in git is like saving, except:

- You can commit changes to multiple files at once, i.e. you save the state of the repository instead of just one file.
- The previous state of the repository is not lost
- When you make a commit, you must add a comment that explains the changes.

Example

git status

Lists the files that have changed (again)

git add, both files

Status now shows us staged file(s) in green

git commit

Status now confirms there are no changes to commit

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   my_zoo.py
        modified:   zoo.py

no changes added to commit (use "git add" and/or "git commit -a")

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git add my_zoo.py zoo.py

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   my_zoo.py
        modified:   zoo.py


simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git commit -m "Add animal noises"
[master db00d66] Add animal noises
 2 files changed, 24 insertions(+), 16 deletions(-)
 rewrite my_zoo.py (68%)

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Staging and committing

Before we can make a commit, there is an interim state called staging.

Only staged changes can be committed.

It is good practice to keep each commit as a single, describable change, e.g. adding a new feature or fixing a bug.

When working on these however, it is very easy to do more than a single change, for example, you spot another bug or fix a typo.

Staging allows us to make these multiple changes to a repository but split these into separate commits.

The command for staging is: **git add**

Example

git status

Lists untracked files

git add, just the file to stage for this commit

Status now shows us staged file(s) in green

git commit

Use –m to add the commit description

(if –m not given a text editor will open)

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        my_zoo.py
        zoo.py

nothing added to commit but untracked files present (use "git add" to track)

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git add zoo.py

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   zoo.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        my_zoo.py


simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git commit -m "Add Animal and Zoo classes"
[master (root-commit) 707eb79] Add Animal and Zoo classes
 1 file changed, 52 insertions(+)
 create mode 100644 zoo.py

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        my_zoo.py

nothing added to commit but untracked files present (use "git add" to track)
```

git log and git show

Git keeps a record of every commit made.

Its tools to interact and manipulate this record is what makes it so powerful.

The first two commands for viewing this record are:

- **git log** – view the entire commit list (for the current branch)
- **git show <commit ID>** – show the changes made in a given commit

Lets view the commits made so far in our example.

git log

Here we see the three commits we've made so far (latest commit at the top, first at the bottom).

The large hash keys are the commit's IDs

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git log
commit db00d66653497ae2fbb134627c1e13a5c176df9c (HEAD -> master)
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:55:58 2022 +0000

    Add animal noises

commit 39c491a0c0b3869ccf643ccb460f36c805db2cfa
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:41:08 2022 +0000

    Create my zoo

commit e8e8ff1e6feeb7bcd3207a9da75c2e205197ef49
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:29:07 2022 +0000

    Add Animal and Zoo classes
```

Example

git show

Notice we specify the commit ID in the git show command.

The commit info and changes made, are then displayed

Note, git show with no commit ID will show the latest commit

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git show db00d66653497ae2fbb134627c1e13a5c176df9c
commit db00d66653497ae2fbb134627c1e13a5c176df9c (HEAD -> master)
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:55:58 2022 +0000

    Add animal noises

diff --git a/my_zoo.py b/my_zoo.py
index ddd43e8..b138e5d 100644
--- a/my_zoo.py
+++ b/my_zoo.py
@@ -8,8 +8,11 @@ from zoo import Animal, Zoo
 best_zoo = Zoo()

 # Create some animals and add to the Zoo
-best_zoo.add_animal(Animal(species="lion", name="Fred", legs=4))
-best_zoo.add_animal(Animal(species="lion", name="Kate", legs=4))
-best_zoo.add_animal(Animal(species="bear", name="Pete", legs=2))
-best_zoo.add_animal(Animal(species="bear", name="Suzy", legs=2))
-best_zoo.add_animal(Animal(species="snake", name="John", legs=0))
+best_zoo.add_animal(Animal(species="lion", name="Fred", legs=4, noise="Roooaaaar"))
+best_zoo.add_animal(Animal(species="lion", name="Kate", legs=4, noise="Rooaar"))
+best_zoo.add_animal(Animal(species="bear", name="Pete", legs=2, noise="Grrrrr"))
+best_zoo.add_animal(Animal(species="bear", name="Suzy", legs=2, noise="Grrrraaar"))
+best_zoo.add_animal(Animal(species="snake", name="John", legs=0, noise="Hisssss"))
+
+for animal in best_zoo.animals:
+    animal.make_noise()
diff --git a/zoo.py b/zoo.py
index 2d826b3..2048271 100644
--- a/zoo.py
+++ b/zoo.py
@@ -3,10 +3,15 @@ Code to define animals and zoos

 """
 class Animal(object):
-    def __init__(self, species, name, legs):
+    def __init__(self, species, name, legs, noise):
         self.species = species
         self.name = name
         self.legs = legs
+        self.noise = noise
+
+    def make_noise(self):
+        print(self.noise)
+        print()


 class Zoo(object):
```

git checkout

This command pops up a few places…

- **git checkout <commit ref>** - move to the state of the given commit.
- **git checkout <branch>** - move to the head of given branch (i.e. the latest commit on that branch).
- **git checkout <file>** - Undo any uncommitted changes to the file (you will loose work). In other words, put the file in the state of the latest commit.

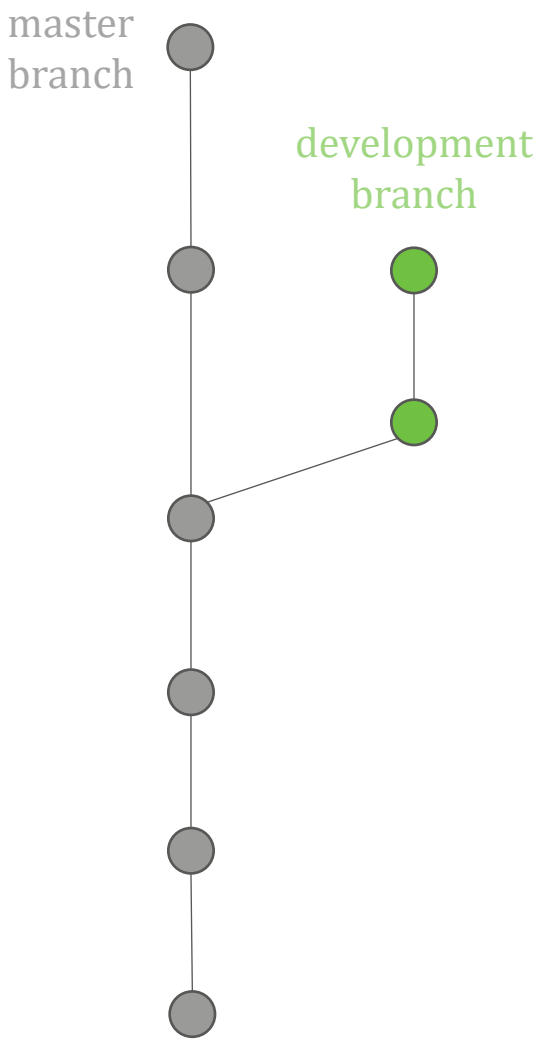
The general meaning of "checkout" is to move files to a committed state.

Branching

In git we can create new branches which contain their own chain of commits.

Branching allows us to develop code in isolation, meaning we cannot accidently break things on the master branch.

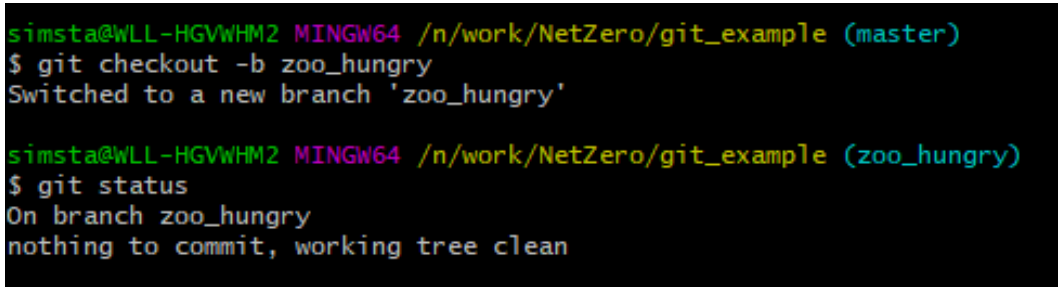
It means we can test and experiment to our hearts content!



Example

git checkout –b zoo_hungry

A git status confirms we are on the new branch



Lets add some commits to this branch…

Example

git log

Here are the three new commits we've made on the zoo_hungry branch

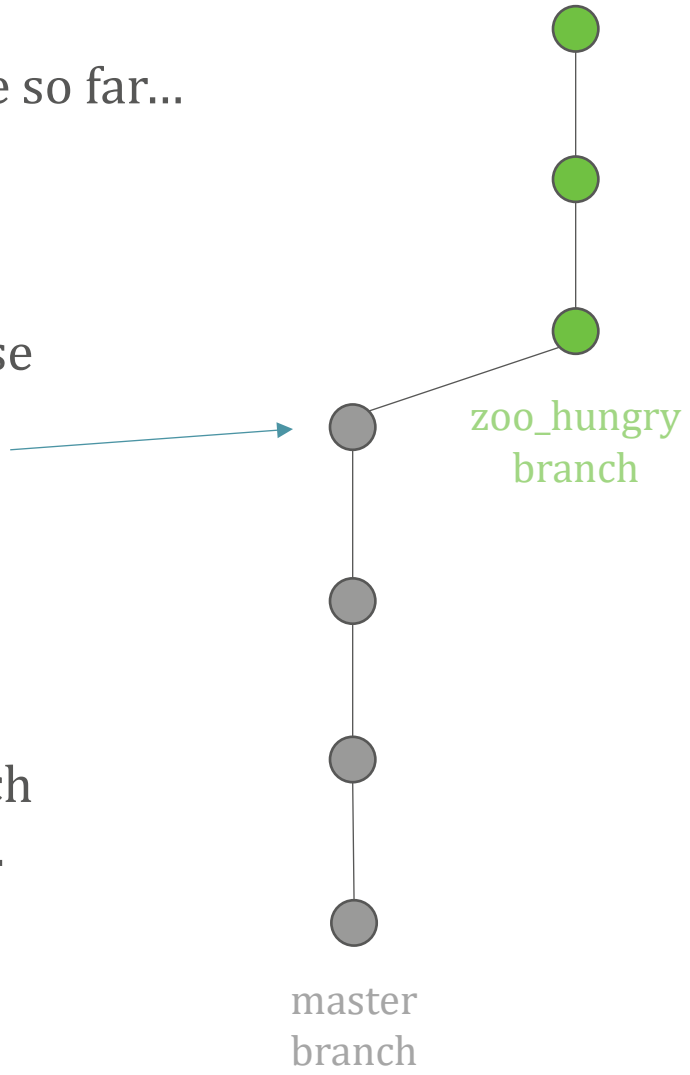
This is where zoo_hungry was branched off master

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (zoo_hungry)
$ git log
commit 1aa9896f93e18f8fc21ff641c6b42cc022546726 (HEAD -> zoo_hungry)
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 15:08:53 2022 +0000

    Print whos hungry in my zoo

commit 1bc6978469640417baa8aa6a2d73bc951ef8c402
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 15:07:01 2022 +0000

    Improve whos_hungry printing

commit b8e2f2b2a19c91a2dec8f1804848ce4cc5819caf
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 15:03:33 2022 +0000

    Add whos_hungry method

commit 3f834fdfa2bba6f6506fab223bfd52f00c21b7ec (master)
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 14:04:03 2022 +0000

    Add animal hunger and feeding

commit db00d66653497ae2fbb134627c1e13a5c176df9c
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:55:58 2022 +0000

    Add animal noises

commit 39c491a0c0b3869ccf643ccb460f36c805db2cfa
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:41:08 2022 +0000

    Create my zoo

commit e8e8ff1e6feeb7bcd3207a9da75c2e205197ef49
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:29:07 2022 +0000

    Add Animal and Zoo classes
```

Example

Here's what we have so far…

Imagine now we are someone else wanting to develop a separate addition to the code from the master branch.



Let's go back to the master branch and create another new branch…

Example

git branch

List all branches and highlight the current branch

git checkout master

Switch (back) to the master branch

git log

Confirms we are back at the same commit where we left it

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (zoo_hungry)
$ git branch
  master
* zoo_hungry

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (zoo_hungry)
$ git checkout master
Switched to branch 'master'

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git log
commit 3f834fdfa2bba6f6506fab223bfd52f00c21b7ec (HEAD -> master)
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 14:04:03 2022 +0000

    Add animal hunger and feeding

commit db00d66653497ae2fbb134627c1e13a5c176df9c
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:55:58 2022 +0000

    Add animal noises

commit 39c491a0c0b3869ccf643ccb460f36c805db2cfa
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:41:08 2022 +0000

    Create my zoo

commit e8e8ff1e6feeb7bcd3207a9da75c2e205197ef49
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:29:07 2022 +0000

    Add Animal and Zoo classes
```

Example

git checkout –b zoo_feed

A git status confirms we are on the new branch →

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git checkout -b zoo_feed
Switched to a new branch 'zoo_feed'

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (zoo_feed)
$ git status
On branch zoo_feed
nothing to commit, working tree clean
```

Lets add some commits to this branch…

git log

The new commit made on the zoo_feed branch

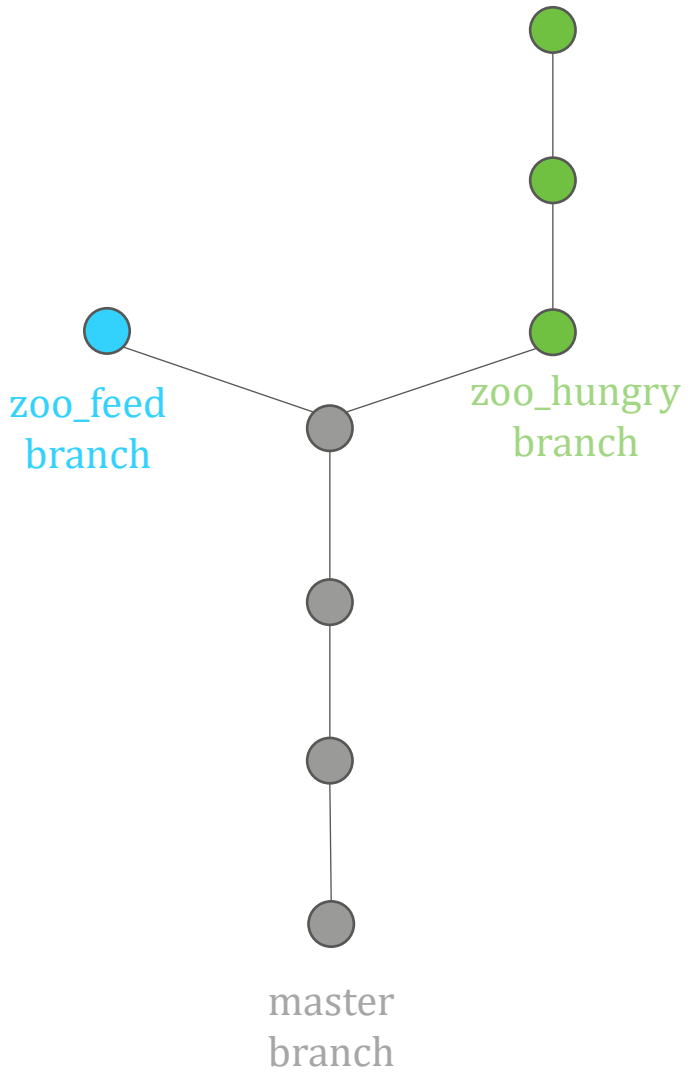
```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (zoo_feed)
$ git log
commit f3b7829ba9628f5a8545ef70f2d23a6beacfd8f5 (HEAD -> zoo_feed)
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 15:40:19 2022 +0000

    Add feed_animals method

commit 3f834fdfa2bba6f6506fab223bfd52f00c21b7ec (master)
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 14:04:03 2022 +0000

    Add animal hunger and feeding

commit db00d66653497ae2fbb134627c1e13a5c176df9c
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:55:58 2022 +0000

    Add animal noises

commit 39c491a0c0b3869ccf643ccb460f36c805db2cfa
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:41:08 2022 +0000

    Create my zoo

commit e8e8ff1e6feeb7bcd3207a9da75c2e205197ef49
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:29:07 2022 +0000

    Add Animal and Zoo classes
```

Example

Now we have three branches all at different points.

What's more, zoo_hungry and zoo_feed have diverged from master in separate ways.

This is where git proves to be very powerful in its ability to **merge** changes…

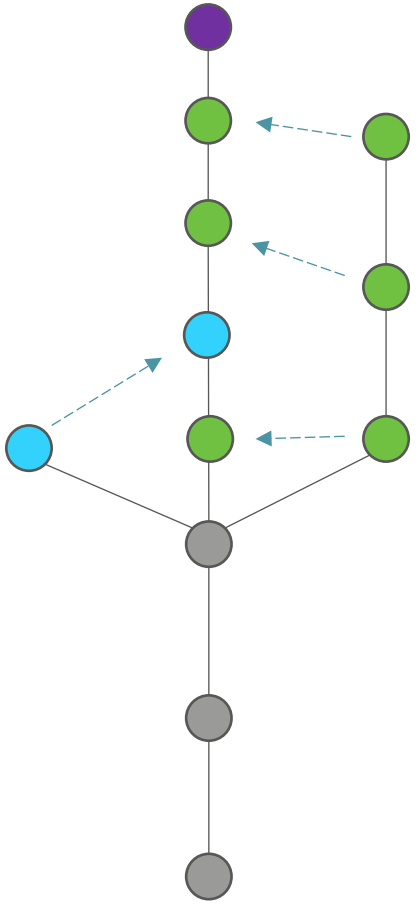


Merging - conflicts

Branching allows us to develop code in isolation and safety.

Once we are happy our developments are correct, we want to merge these back onto the master branch.

If changes are made to the same code on the branch being merged, there will be a conflict…



Example

git checkout master

Must be on branch we want to merge into.

git merge zoo_hungry

As master has not changed since branching off zoo_hungry, this is a fast forward merge

git log

We see the commits from zoo_hungry now on the master branch

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (zoo_feed)
$ git checkout master
Switched to branch 'master'

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git branch
* master
  zoo_feed
  zoo_hungry

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git merge zoo_hungry
Updating 3f834fd..1aa9896
Fast-forward
 my_zoo.py |  2 ++
 zoo.py    | 11 ++++++++++
 2 files changed, 13 insertions(+)

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git log
commit 1aa9896f93e18f8fc21ff641c6b42cc022546726 (HEAD -> master, zoo_hungry)
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 15:08:53 2022 +0000

    Print whos hungry in my zoo

commit 1bc6978469640417baa8aa6a2d73bc951ef8c402
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 15:07:01 2022 +0000

    Improve whos_hungry printing

commit b8e2f2b2a19c91a2dec8f1804848ce4cc5819caf
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 15:03:33 2022 +0000

    Add whos_hungry method

commit 3f834fdfa2bba6f6506fab223bfd52f00c21b7ec
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 14:04:03 2022 +0000

    Add animal hunger and feeding

commit db00d66653497ae2fbb134627c1e13a5c176df9c
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Mon Nov 7 17:55:58 2022 +0000

    Add animal noises

commit 39c491a0c0b3869ccf643ccb460f36c805db2cfa
Author: Simon Stanley <simsta@ceh.ac.uk>
```

git merge zoo_feed

Because we merged zoo_hungry, master has now changed since zoo_feed was branched off.

git status gives us some guidence

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git merge zoo_feed
Auto-merging zoo.py
CONFLICT (content): Merge conflict in zoo.py
Auto-merging my_zoo.py
CONFLICT (content): Merge conflict in my_zoo.py
Automatic merge failed; fix conflicts and then commit the result.

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   my_zoo.py
        both modified:   zoo.py

no changes added to commit (use "git add" and/or "git commit -a")
```

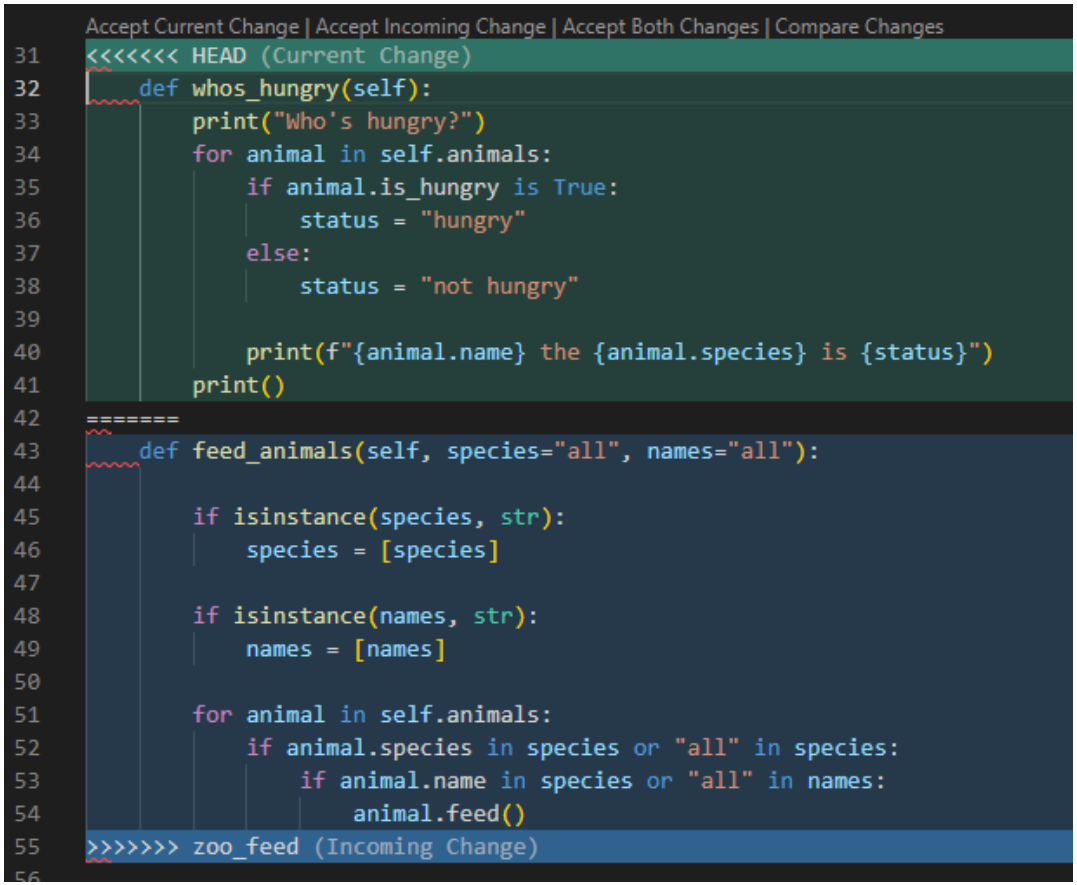
This means git will attempt to auto merge. However, in both branches we made changes to the same part of the file, and so there is a conflict!

These must be manually resolved…

Example

What's in the current branch (master)

What's in the branch being merged in (zoo_feed)



In this case we want to keep both changes.

Note that we can edit these files how we like at this point as this all becomes part of the merge commit.

Example

Add and commit changes once conflicts are resolved. (No need to change the default commit message)

git log shows us the all the commits from both branches now there, plus the merge commit at the top.

```
simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master|MERGING)
$ git add my_zoo.py zoo.py

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master|MERGING)
$ git commit
[master f076f46] Merge branch 'zoo_feed'

simsta@WLL-HGVWHM2 MINGW64 /n/work/NetZero/git_example (master)
$ git log
commit f076f4625de39ca056e68693e1bb914dc663d341 (HEAD -> master)
Merge: 1aa9896 f3b7829
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 17:48:14 2022 +0000

    Merge branch 'zoo_feed'

commit f3b7829ba9628f5a8545ef70f2d23a6beacfd8f5 (zoo_feed)
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 15:40:19 2022 +0000

    Add feed_animals method

commit 1aa9896f93e18f8fc21ff641c6b42cc022546726 (zoo_hungry)
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 15:08:53 2022 +0000

    Print whos hungry in my zoo

commit 1bc6978469640417baa8aa6a2d73bc951ef8c402
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 15:07:01 2022 +0000

    Improve whos_hungry printing

commit b8e2f2b2a19c91a2dec8f1804848ce4cc5819caf
Author: Simon Stanley <simsta@ceh.ac.uk>
Date:   Tue Nov 8 15:03:33 2022 +0000

    Add whos_hungry method

commit 3f834fdfa2bba6f6506fab223bfd52f00c21b7ec
```