GitHub

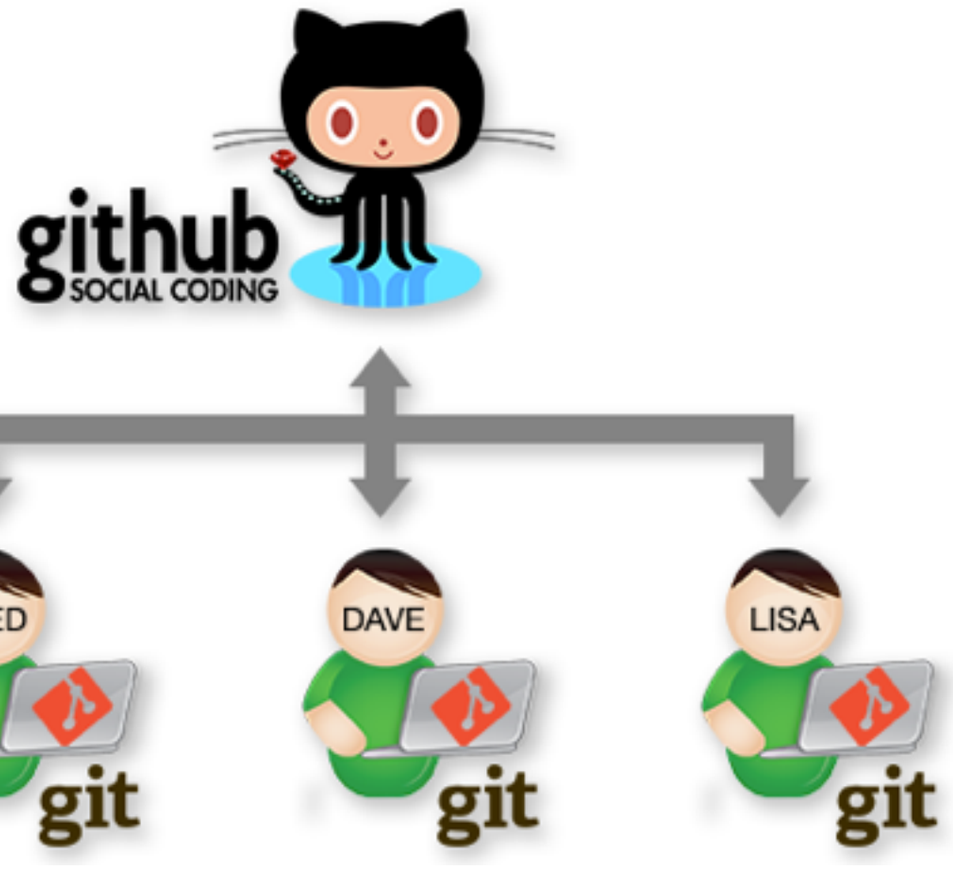
What is GitHub?

A repository for an open source, version control system - where developers can store projects and network with others

Allows for distributed, collaborative development

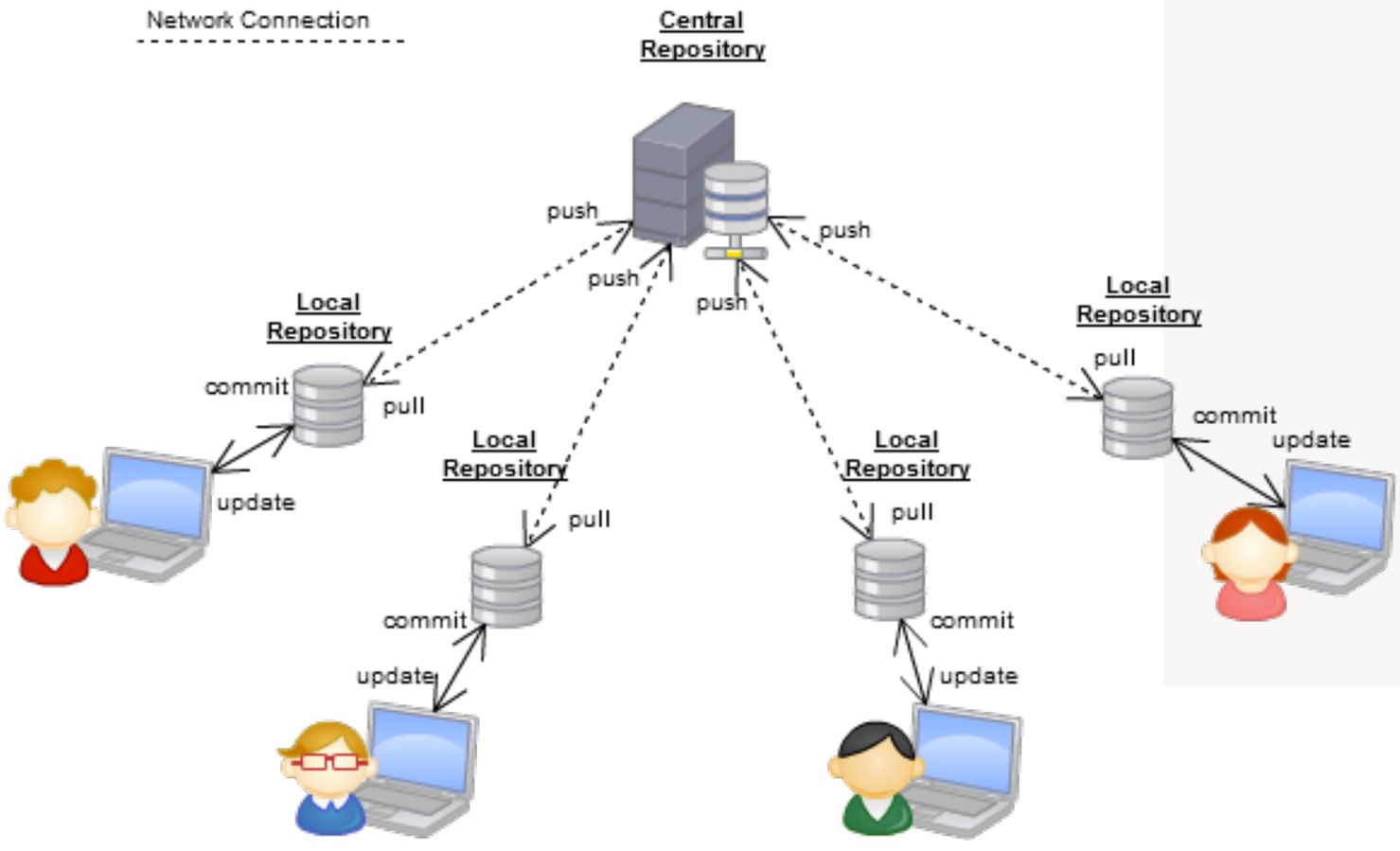
Manages and stores revisions to projects;

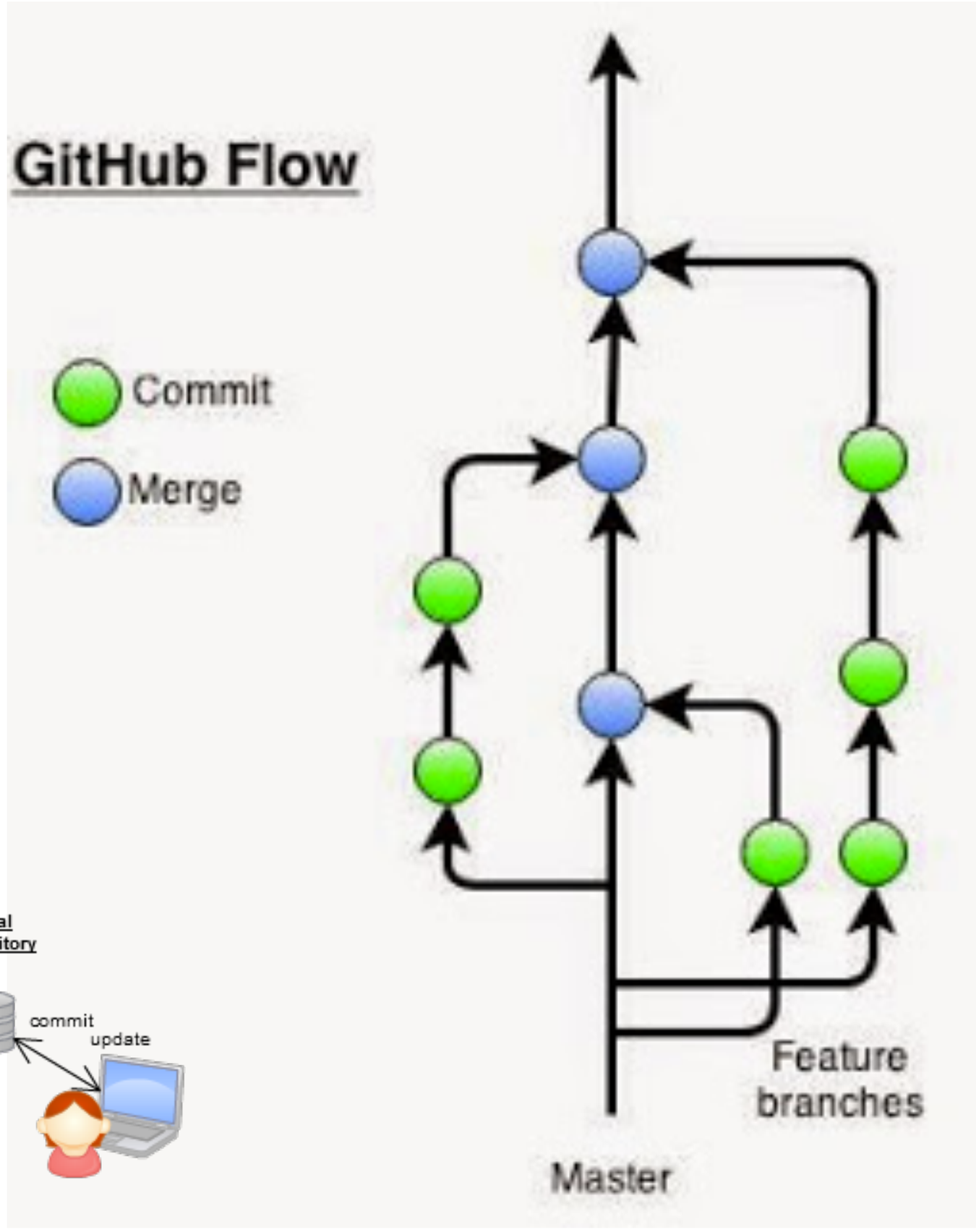
Projects can be code, documents, data
Rmarkdown…pretty much anything

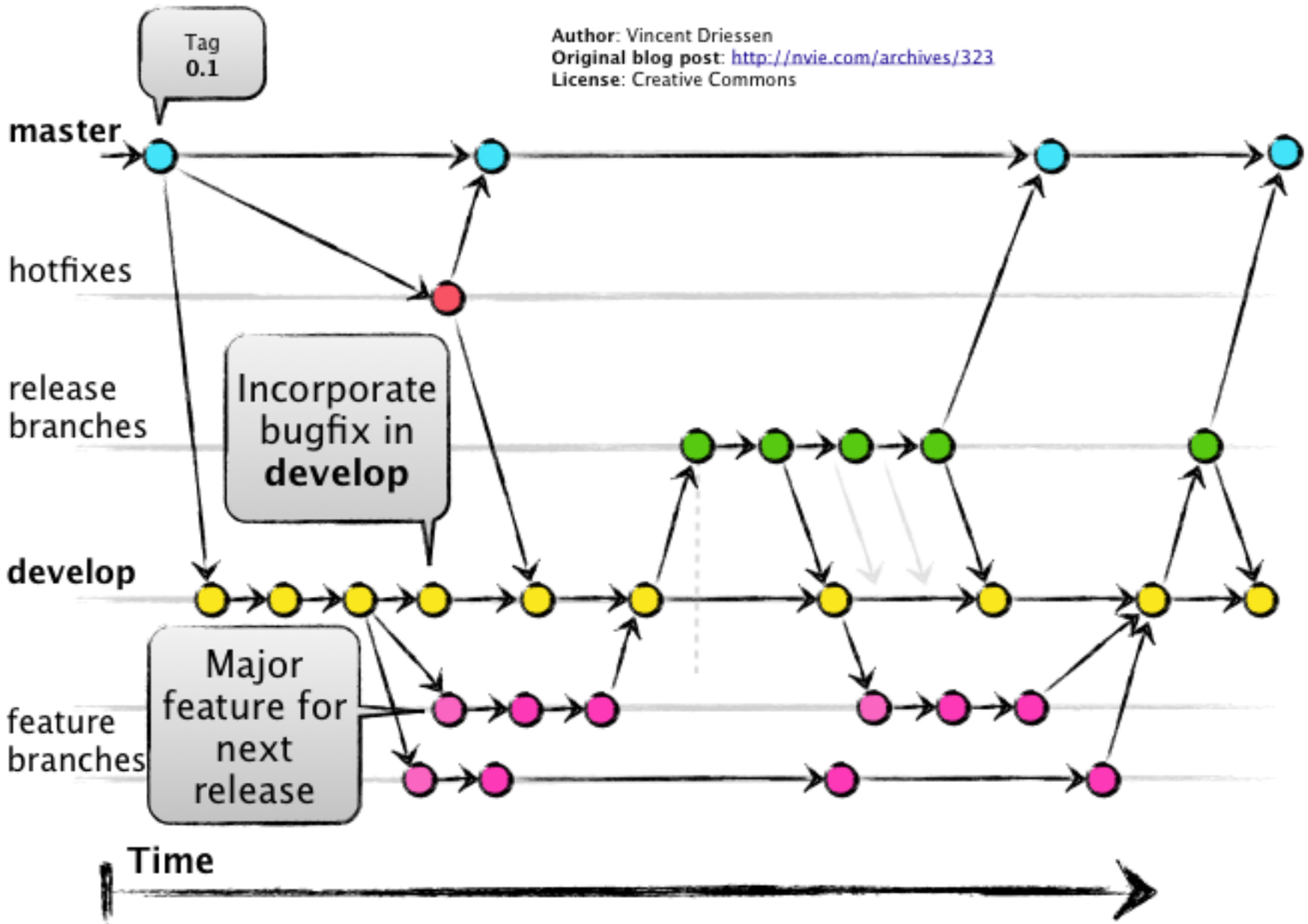


Why use version control?

- Keep track of your own change to code
- Efficient updating, error tracking
- Multiple people working on a project
- User A makes changes to a particular part of the project
- User B also makes changes to the same part of the project
- Git allows both user A and user B to upload their revisions without them overwriting one another
- Both revisions can be merged together without losing work from either






Tag
0.1
Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323
License: Creative Commons
master
hotfixes
release
branches
Incorporate
bugfix in
develop
develop
Major
feature for
next
release
feature
branches
Time

1. Create GitHub user account on GitHub.com

2. Download and install the Git program on your local computer

3. Configure Git with your user information
   - User name – this can be any name you want, it will be used to credit your contributions to a project
   - Use the email you used to sign up for your GitHub account

4. Create or Clone a Git repository from GitHub to your local computer

follow commands on
https://help.github.com/articles/create-a-repo/

Version Control

1. install git on your system

- You can now use Git with any code or data that you have
  - Rstudio makes git even easier to use
  - In Rstudio - Git repositories are organized by Rstudio Projects
  - You can put this project into a directory that is already under version control for some other reason (and use the Url for version control repository)
  - Or you can start from scratch - creating a new repository

Version control in R studio

- For Rstudio -
  - Rstudio provides an interface for common git commands
  - You can also use a shell command in R to use other, less common git command when you need them
  - We will start with using git in Rstudio; because its easy but then move to the bigger ‘git’ word

Git and Rstudio

- Download Git
- Open Rstudio
  - got to: tools:global options
    - set path to git executable
      - to find that from the command line ("which git")
- Create a new project

Git and Rstudio

- use **git:commit** to enter your project to start
- make a change to your Rmarkdown file
- use **git:diff** to see what you've changed

- when you are happy with change…stage the changes (click); use **git:commit** again to commit your new code
- if your not happy with your new code…use **git:revert** to go back to what you had before you started mucking about

Git and Rstudio

- commit your climate analysis code and tests
- make a change to the code, test it, commit it
- add a new test to your code
- add some error checking to that function to make sure that input data are “realistic”
- write a new function that will read some climate data from a file (have the file name be input to you function); call your spring climate analysis function and the uses the results to generate a new climate data sequence of daily values for the year with the wettest spring

Git and Rstudio

- what if you did something silly - like committed something that you shouldn't have: Can we go back? yes…but we need to go to "real" git, by going to the shell
- in the shell we access git commands as
  - *git commit*
  - *git revert*
  - *git log* (shows you the changes you've made)
  - *git diff* (shows you how the current files (not committed yet) are different
  - *git help* (shows all git command)
  - *git help* command (show help for that git command)

Git Commands:

All branches associated with a project are pulled down to your local computer.
To see what branches are locally available:

git branch

The active branch you are on will be denoted by an asterick *

To see all branches, including remote branches:

git branch –all

To switch to a particular branch:

git checkout branch_name

To create your own feature branch off of an existing branch:

git checkout –b myfeaturebranchname existingbranchname

i.e.

git checkout –b MyFeature develop

This creates a new branch called MyFeature off of the existing develop branch, and switches you into this new branch.
A feature branch is meant to exist in your local repository, on your computer

Git commands:

To add a file to your local repository:

git add file_name

To change the name of a file (move):

git mv

To delete files (remove):

git rm

Check the status or your repository – this will list what files have been modified since the last commit, and list any files not currently under version control (i.e. new files you have created:

git status

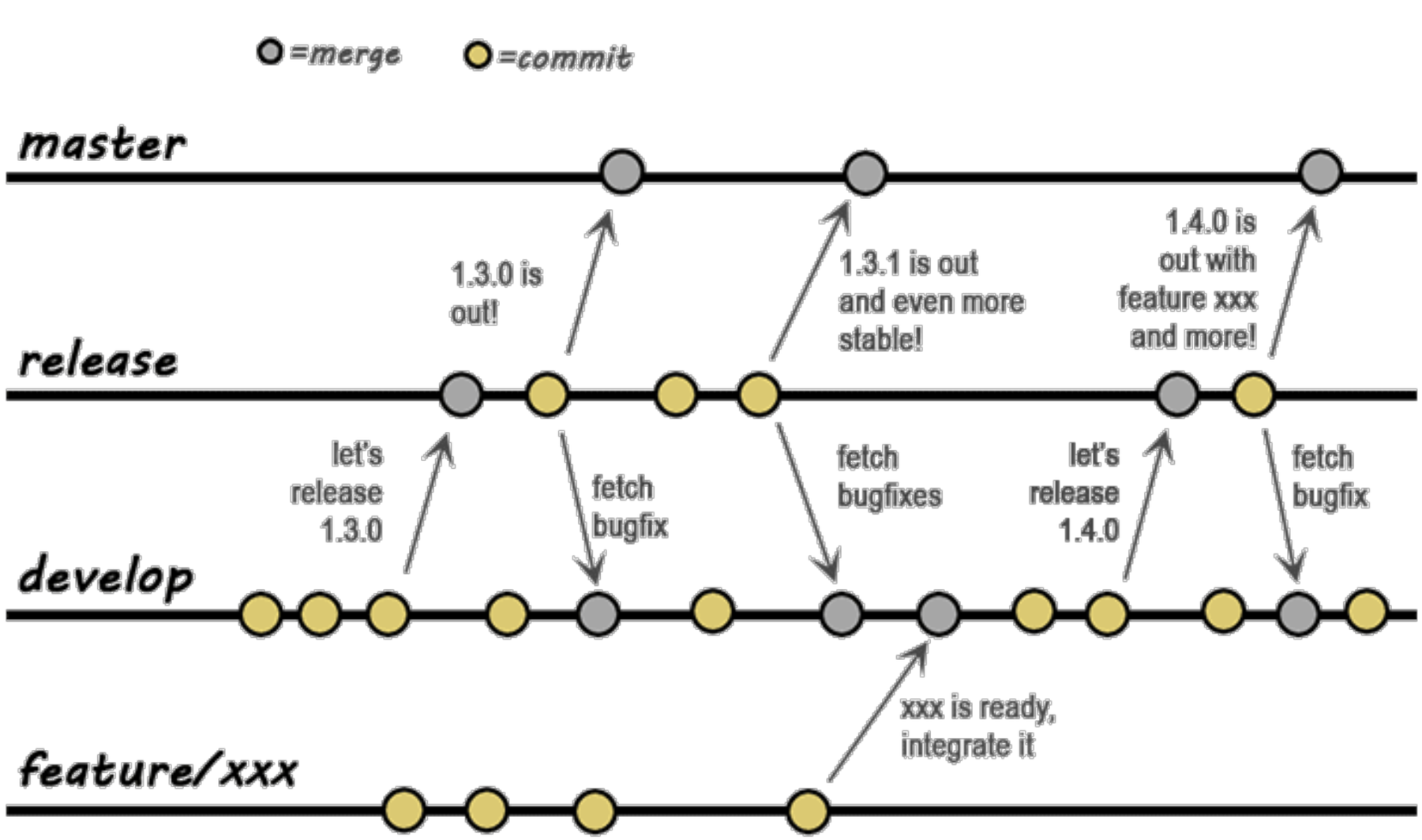
Git and Rstudio

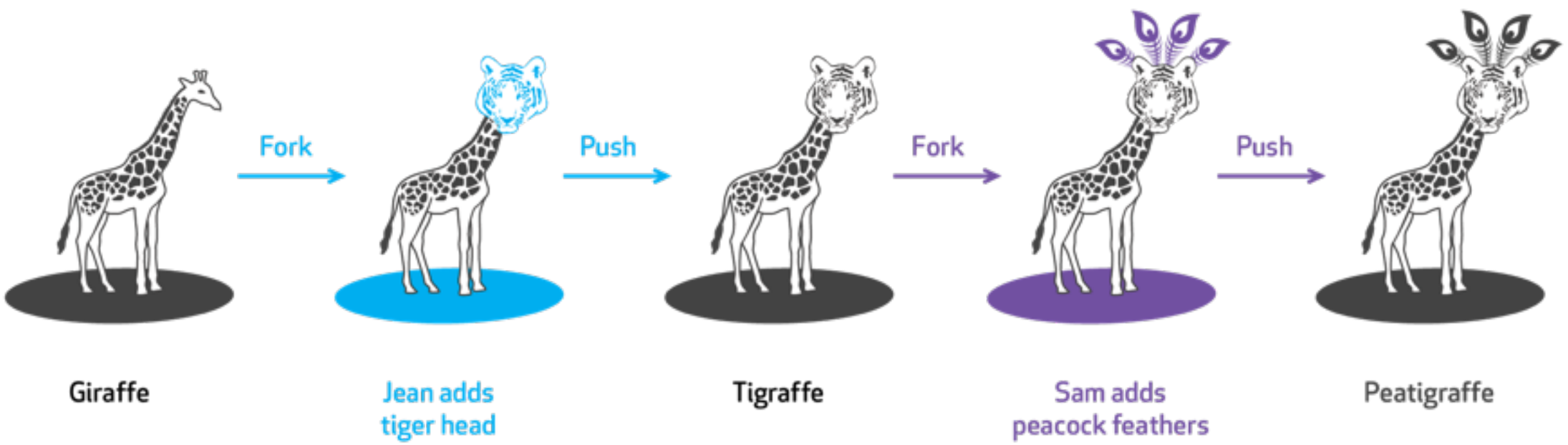
- now we can use git log to see the revision # of the code prior to our last commit (when we mucked things up)
- we can use git checkout # to get this back
- and then commit that so that "old" version now becomes the "HEAD" the current code

- *git log —reverse*
  - the — is an argument for git log
  - *git help log* …shows all possible arguments
- note the commit number!
- now we can get it back using *git checkout #*

Git and Rstudio

- now we can get it back using *git checkout #*
- so commit # is now the HEAD (the current version)
  - what we see …look at git:history to see
- if we want to keep this version, we create a new branch
    - *git checkout -b revised*
- OK What about branching….lets say we want a version to do some experimentation…some development


=merge
=commit
master
release
develop
feature/xxx
1.3.0 is out!
1.3.1 is out and even more stable!
1.4.0 is out with feature xxx and more!
let's release 1.3.0
fetch bugfix
fetch bugfixes
let's release 1.4.0
fetch bugfix
xxx is ready, integrate it


Fork
Push
Fork
Push
Giraffe
Jean adds tiger head
Tigraffe
Sam adds peacock feathers
Peatigraffe

Git and Rstudio

- We can create a new branch to do experimental development
  - create a new branch (whose parent is the current branch *"master"*, call it *"experimental"*
  - to switch between branches '*git checkout branchname*'
  - so 1. *checkout master 2. checkout -b experimental*

```
#' Read a climate input file and convert to monthly or year time step
#'
#'
#' @param filename
#'  file must have tmax, tmin, rain (precip in mm), year, month (integer), day
#' #' @param timestep; must be "m" or "y" or "d"; default is "m
#' @return dataframe with monthly results

generate_monthly_clim = function(filename, timestep="m") {
  dataset = read.table(filename, header=T)

  if (timestep=="m") {
  newdata = aggregate(dataset[,c("tmax","tmin","month")], by=list(dataset$month), mean)
  tmp = aggregate(dataset[,c("rain","month")], by=list(dataset$month),sum)
  newdata=tmp$rain
}
  return(newdata)
}
```

Git and Rstudio: MERGING

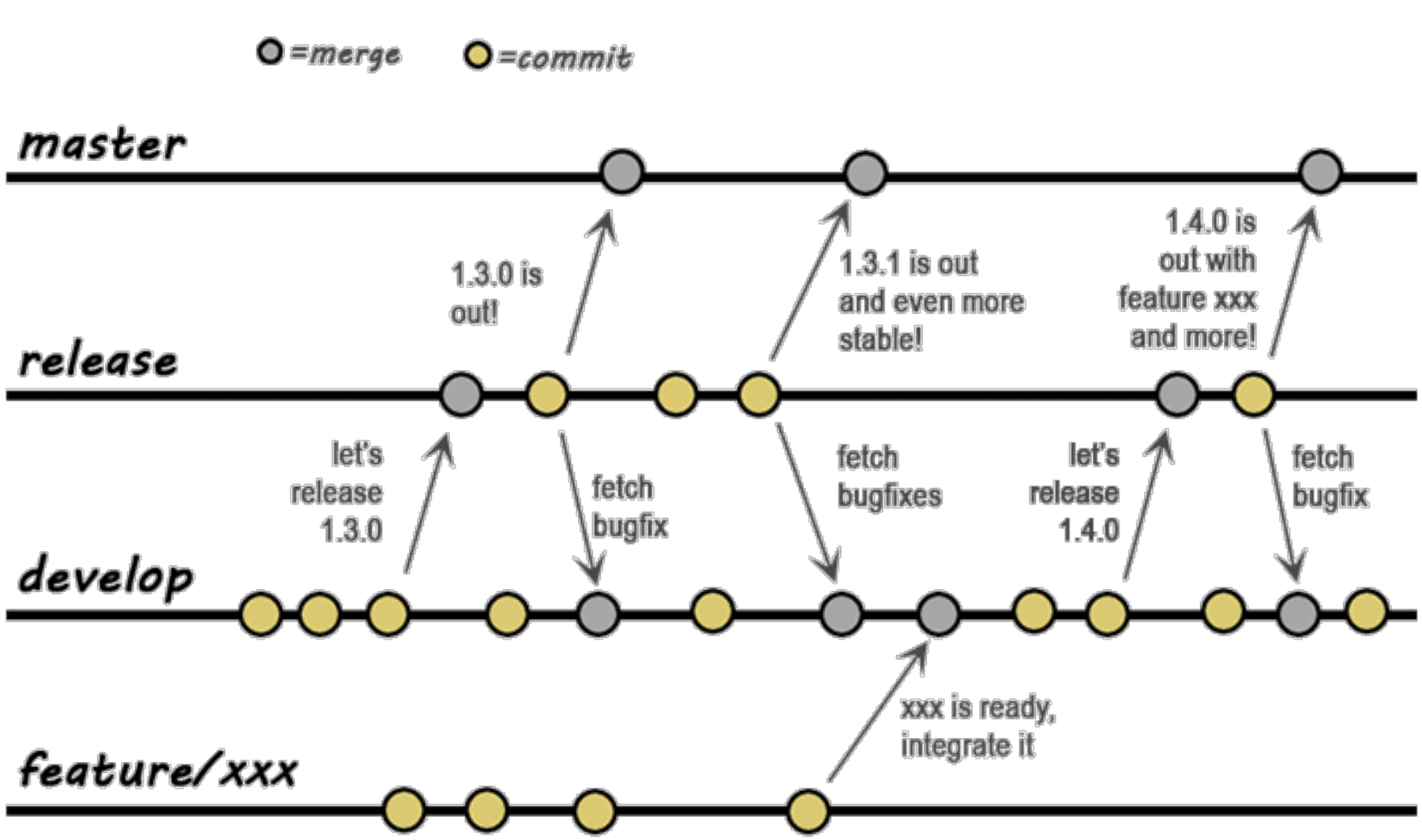
- we can now do git:commits on our new *experimental* branch
- make sure you are on the "*experimental*" branch in Rstudio

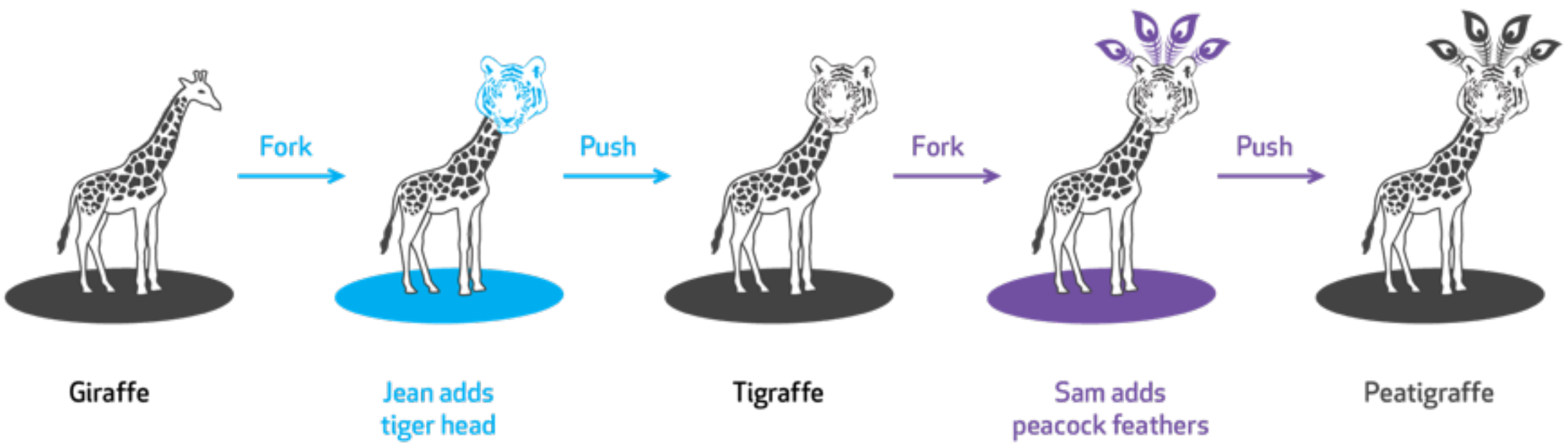
Git and Rstudio: MERGING

- Ok lets say we want merge our corrections: in revised with our master branch
- Convention uses the master branch as the “releasable” or main version of the code

Git and Rstudio: MERGING

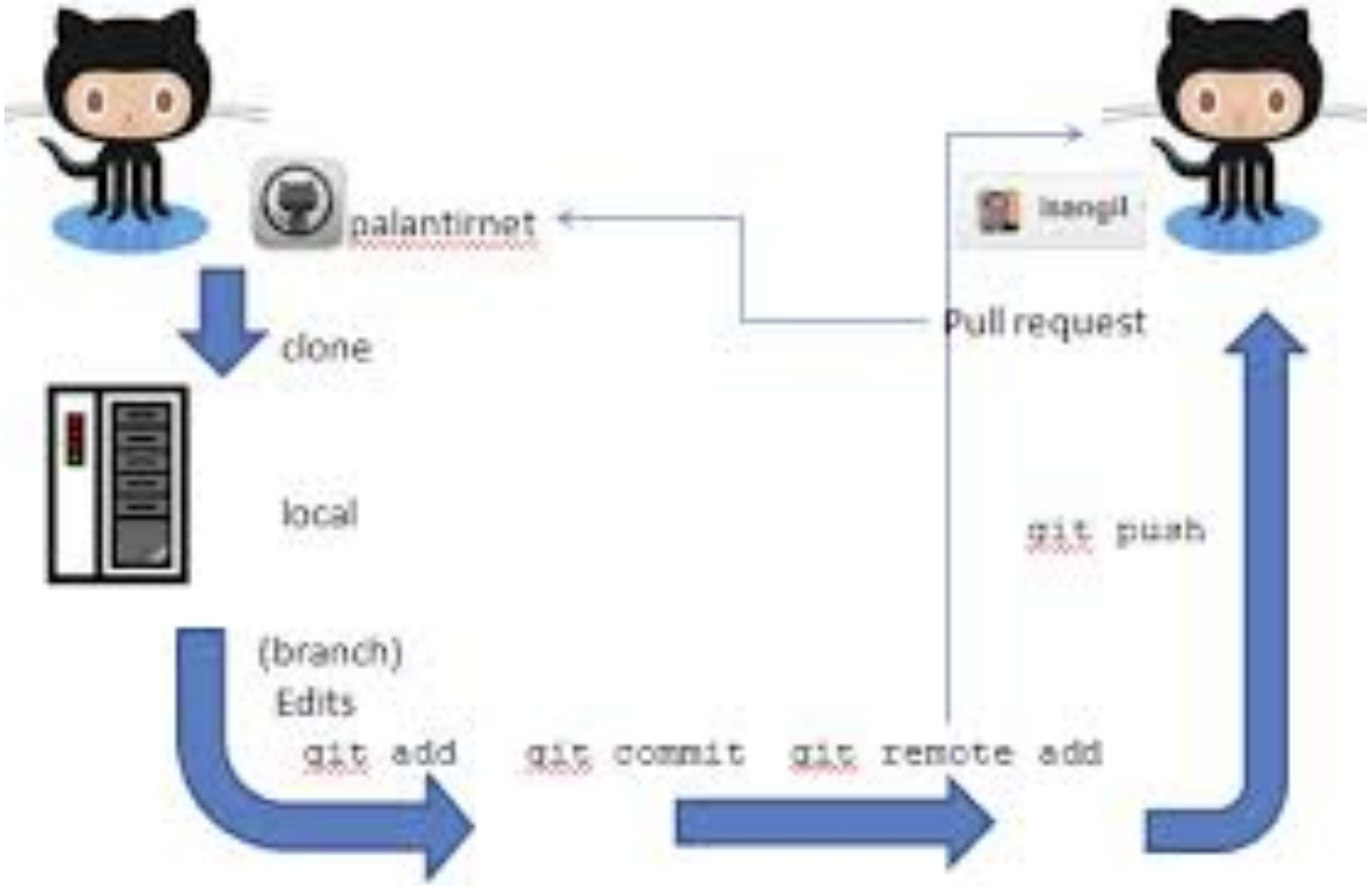
- Ok lets say we; we want merge new modifications: into a revised into our master branch
- Convention uses the master branch as the "releasable" or main version of the code
- To merge "experimental" into "master"
  - set "master" as the current branch (*git checkout*)
  - use *git merge revised*

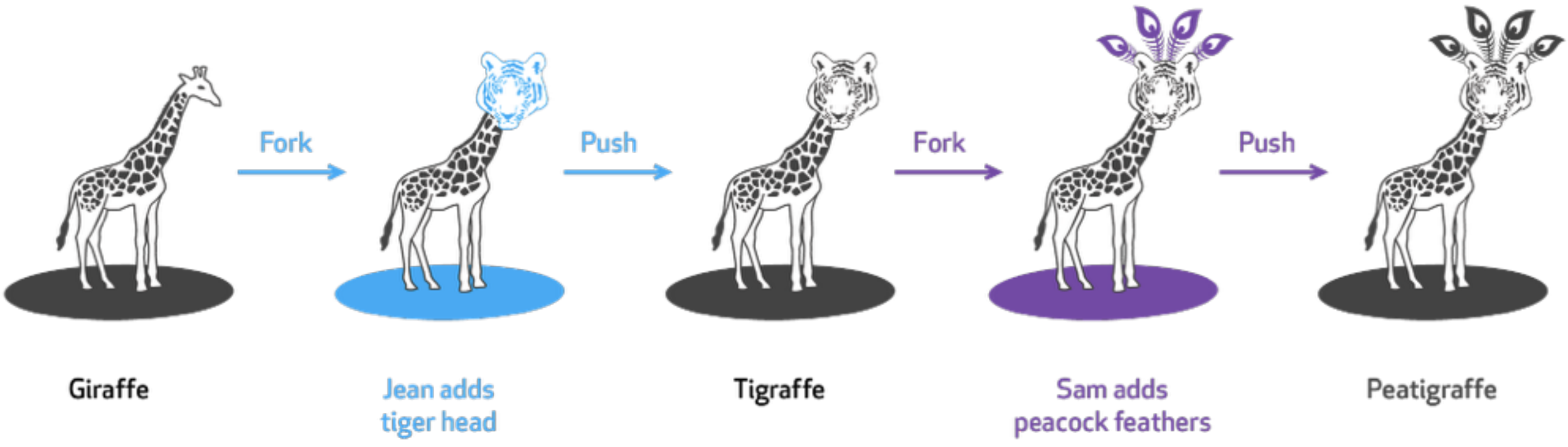

=merge
=commit
master
release
develop
feature/xxx
1.3.0 is out!
1.3.1 is out and even more stable!
1.4.0 is out with feature xxx and more!
let's release 1.3.0
fetch bugfix
fetch bugfixes
let's release 1.4.0
fetch bugfix
xxx is ready, integrate it


Fork
Push
Fork
Push
Giraffe
Jean adds tiger head
Tigraffe
Sam adds peacock feathers
Peatigraffe

Code Development

- LOCAL
- Design/Revise your branch
- Test
- Commit to your branch
- Merge your branch with master (or other main branch)
- LINK TO SHARED GIT REPOSITORY
    - Push - add your updates to remote repository
    - Pull - gets other peoples updates to your local repository


palantirnet
isangil
clone
Pull request
local
git push
(branch)
Edits
git add
git commit
git remote add


Fork
Push
Fork
Push
Giraffe
Jean adds
tiger head
Tigraffe
Sam adds
peacock feathers
Peatigraffe

Git Commands

Git commands begin with *git*

Download code from GitHub by *cloning* the repository.

For example, if you were going to download RHESSys source code from the Git repository hosted on GitHub:

git clone https://github.com/RHESSys/RHESSys.git rhessys_git

This would ‘copy’ the RHESSys source code hosted at that web address into a directory on your local computer called rhessys_git

When you run git clone, every file for the history of the project is pulled down by default – and it automatically creates a remote connection called origin pointing back to the original repository

Git commands:

To merge your feature branch with a local branch on your computer, you must first activate the branch you want to merge your feature branch with:

git checkout master

Now merge your feature branch with the develop branch:

git merge MyFeature

To delete your feature branch after merging it”

git branch –d MyFeature

To contribute the code back the repository hosted on GitHub:

git push origin develop

To update your local repository to reflect changes that may have been made since you last checked it out:

git pull

Assignment

- Building on your function that analyzes spring climate
- Write two additional functions to
  - read in daily climate data
  - write out a new climate data file that combines daily temperature from the coldest spring with daily precipitation from the wettest spring; allow the user to specify whether output pseudo climate date is daily, monthly or yearly
  - for each new routine - write at least 3 tests
- Put all of this on github as a new branch from your original code
- make sure your repository is public - and just submit the URL of the repository on Gauchospace