Jumping in

- We are going to use R; but the basic design of programs (models) are similar across many programming languages
- Why R?
  - Free (and open source) software
  - Good (and getting better) visualization tools
  - Growing user community who make their R code available
    - (currently 2800+ user packages on CRAN R server)
  - Links with other tools and languages (GIS, Python, C, C++…)
  - Built in tools to deal with space and time
  - Lots of user support

R …

- Why not R?
  - Not particularly computationally efficient (e.g slow for repetitive computations) ; hard to parallelize
  - Not the right tool for developing really complex models (you don't develop GCMs in R!)

Useful R Websites

- http://www.r-project.org/ (Main R site)
- http://www.revolutionanalytics.com/ (Commercial version of R, but lots of free stuff
- http://spatial.ly/r/ (Using R with Spatial Data)
- http://cran.r-project.org/doc/contrib/Short-refcard.pdf (really useful reference card)
- http://www.rdocumentation.org/ (a searchable database of R libraries)

- There are many R tutorials out there - feel free to post your favorites on Gauchospace

Reproducibility

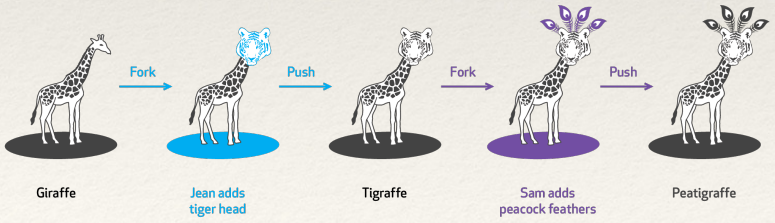
- Start with tools that help you to organize your work
  - Keep track of changes that you make as you go (think of track changes in "Word" GIT
  - Support collaboration and sharing with others GIT
  - Allow you to combine different tools, data formats, output formats

What is GIT

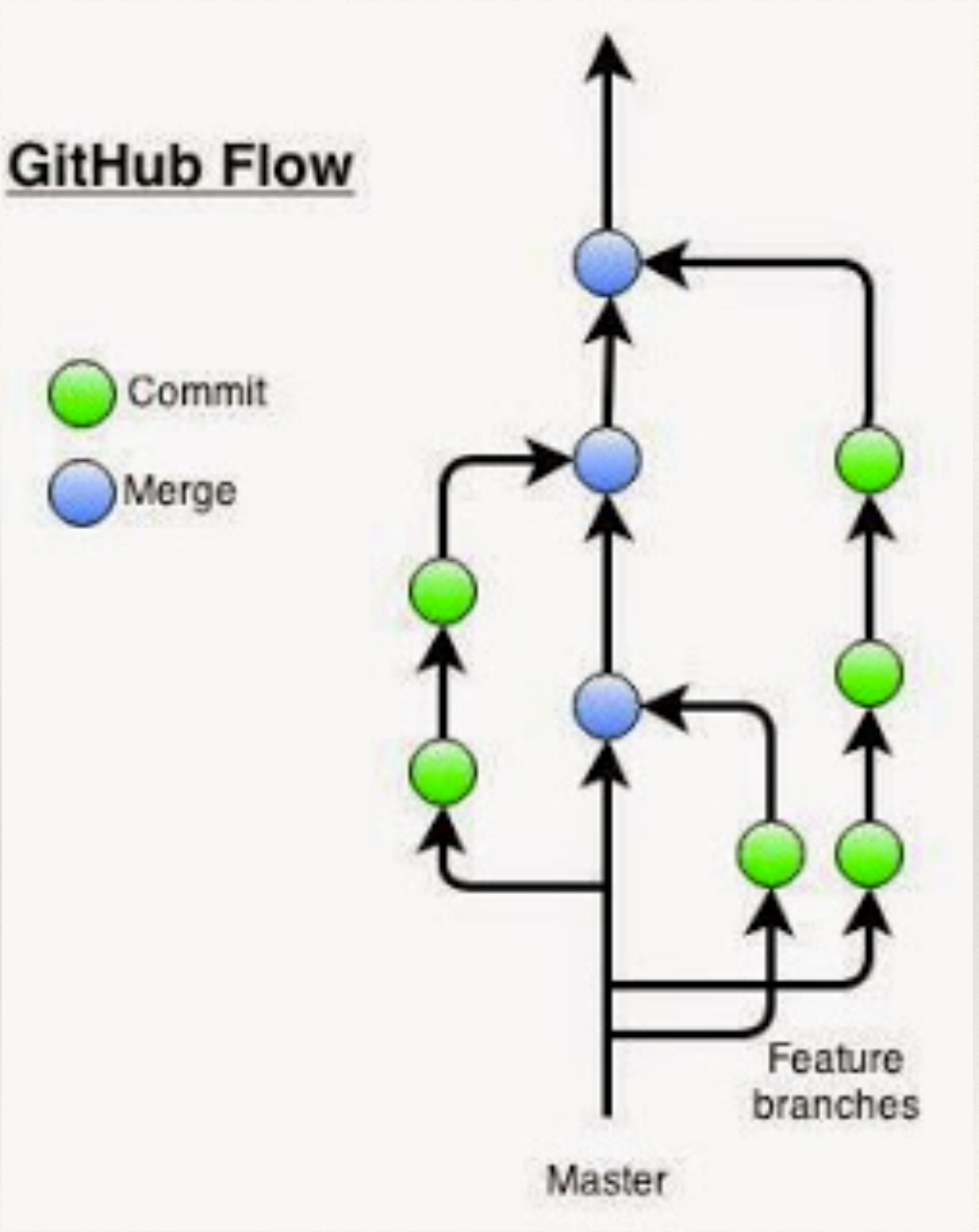
- Version controls system: a way of keeping track of changes to *work* that evolves through time
  - *work* can be data, programs, documents…anything
  - allows you to see what has changed and go back to old versions if need - “back in time”
  - facilitate collaboration - manages multiple people working on the same thing

Why use version control?

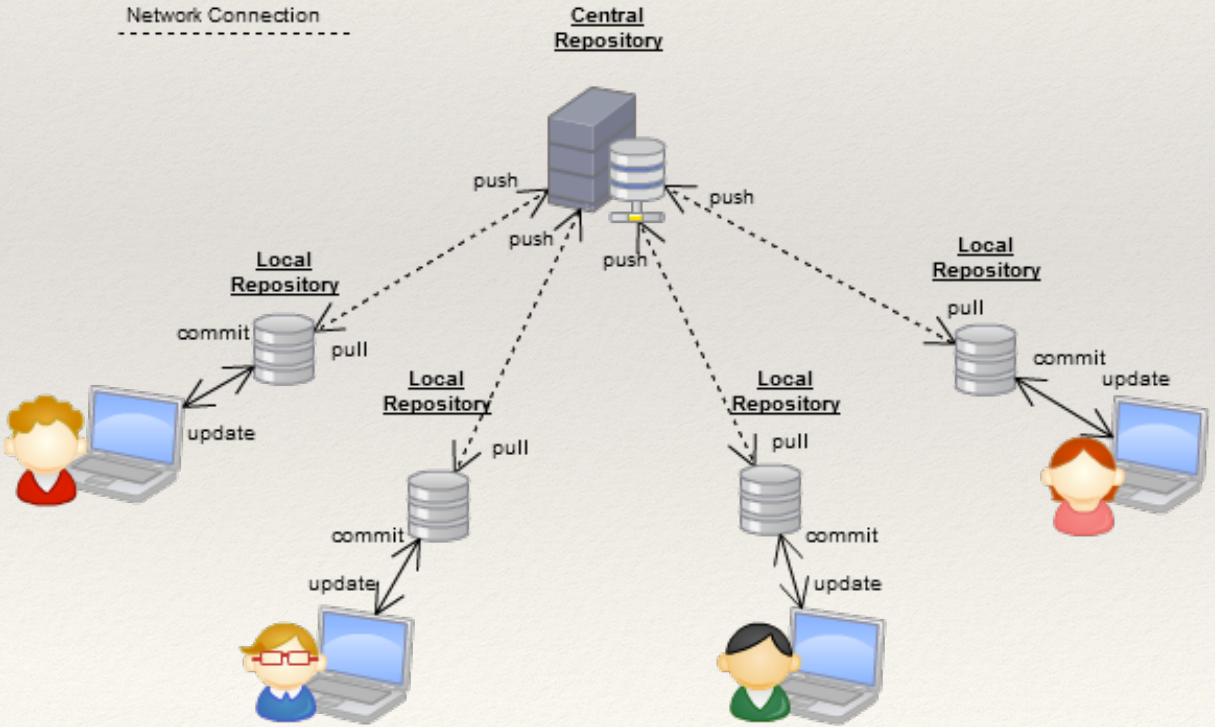
- Keep track of your own change to code
- Efficient updating, error tracking
- Multiple people working on a project
- User A makes changes to a particular part of the project
- User B also makes changes to the same part of the project
- Git allows both user A and user B to upload their revisions without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



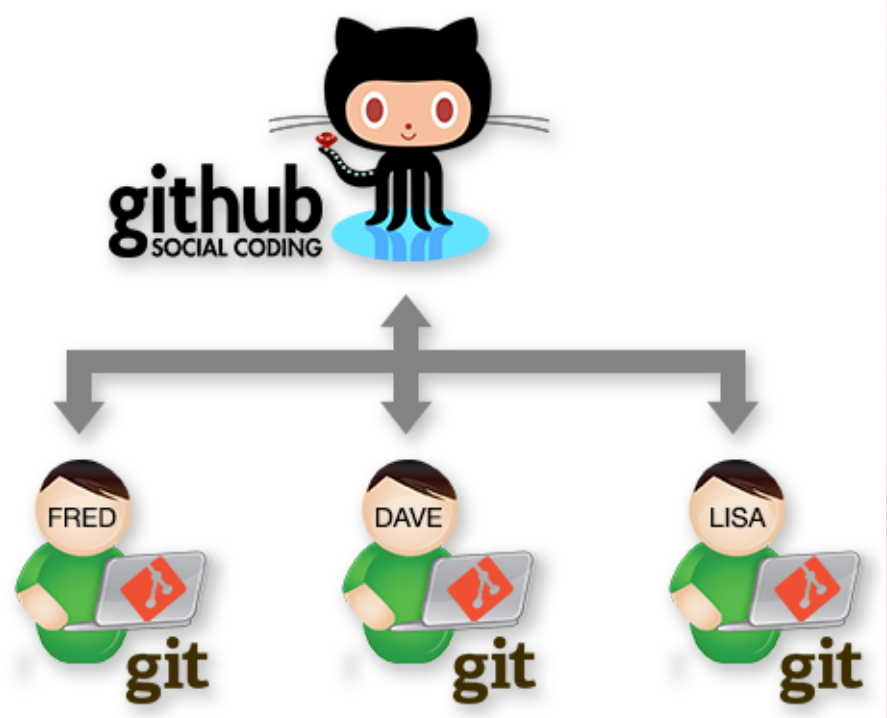
peacock feather, tiger, giraffe from The Noun Project



git and github work together



git and github work together



- Git is local (on your machine) and keeps track of your work
- Github is on the web (it allows you to share your work with others)
- Rstudio supports the use of Git - and linking your Git repository with the github repository
- BUT Git can be used outside of R! its much bigger and can be used with many programs - operating system commands

GitHub

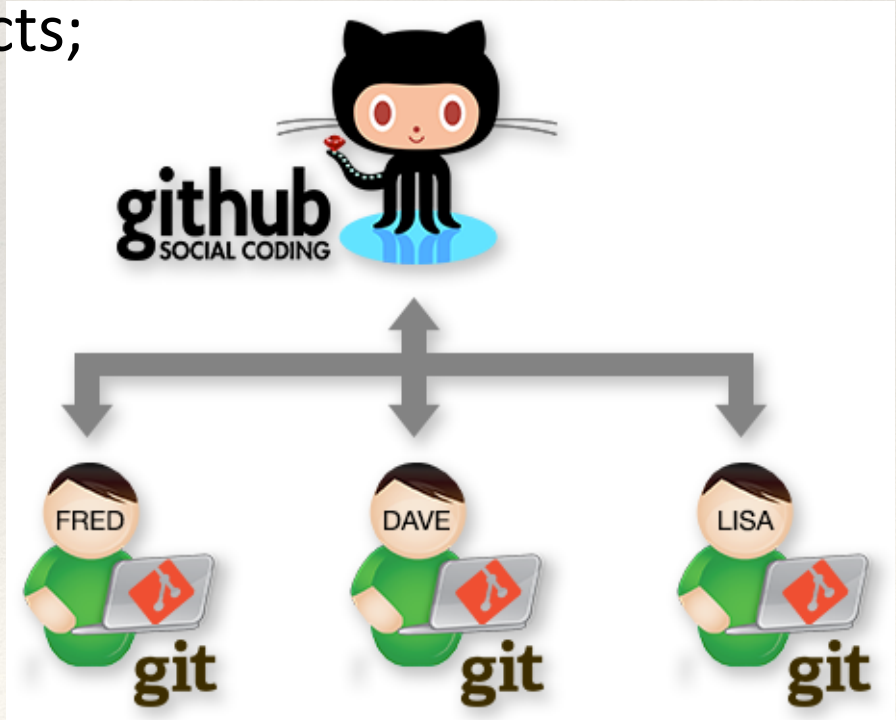
What is GitHub?

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

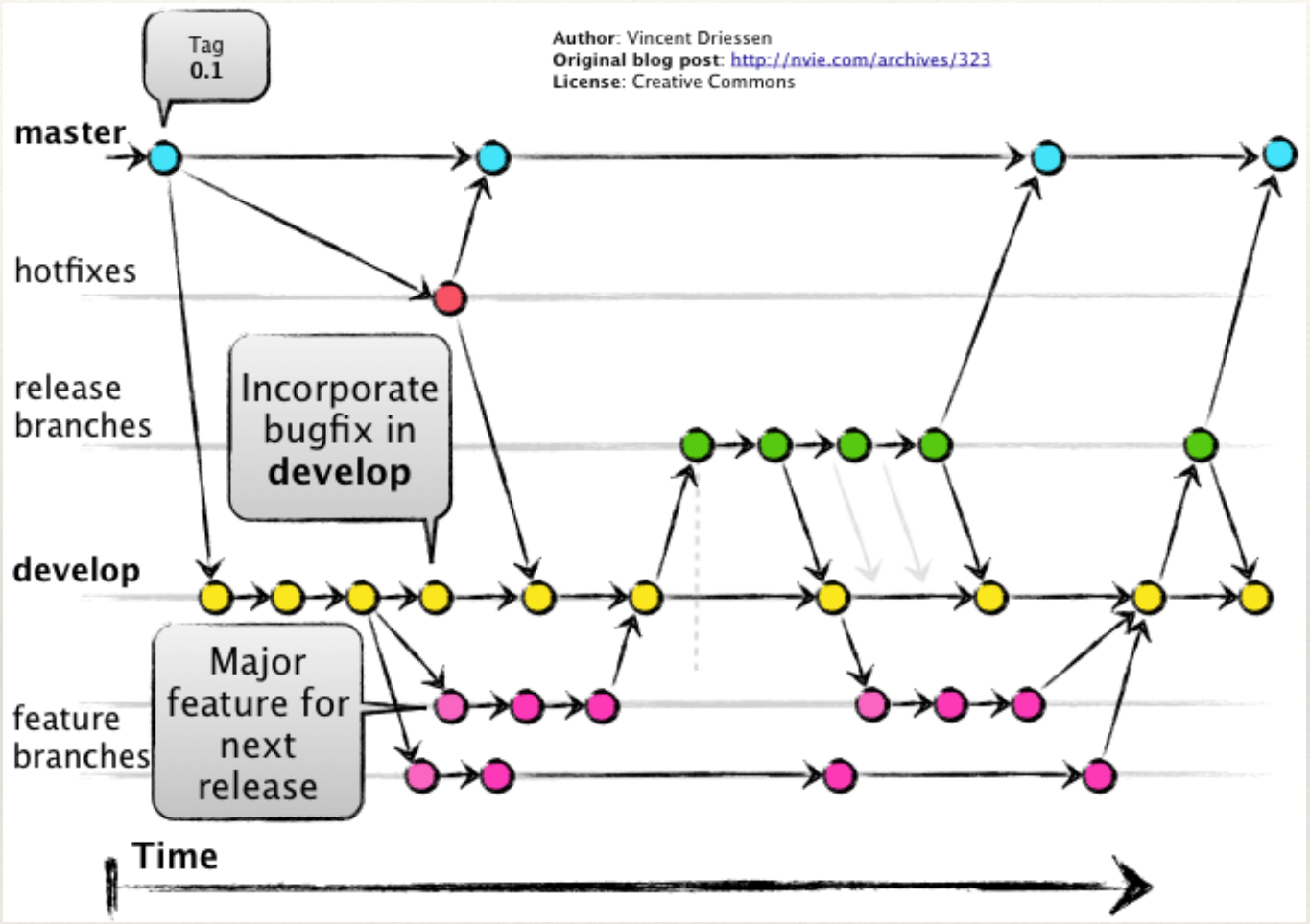
Allows for distributed, collaborative development

Manages and stores revisions to projects;

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



A software developers view of Git



How you set things up

- You can now use Git with any code, data, documents
  - Can work with Git directly from github.com; or from shell commands on unix based operating systems..or..
  - Rstudio makes git easier to use
  - In Rstudio - Git repositories are organized by Rstudio Projects
  - You can put any project into a directory that is already under version control for some other reason (and use the Url for an existing repository)
  - Or you can start from scratch - creating a new repository for your project

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

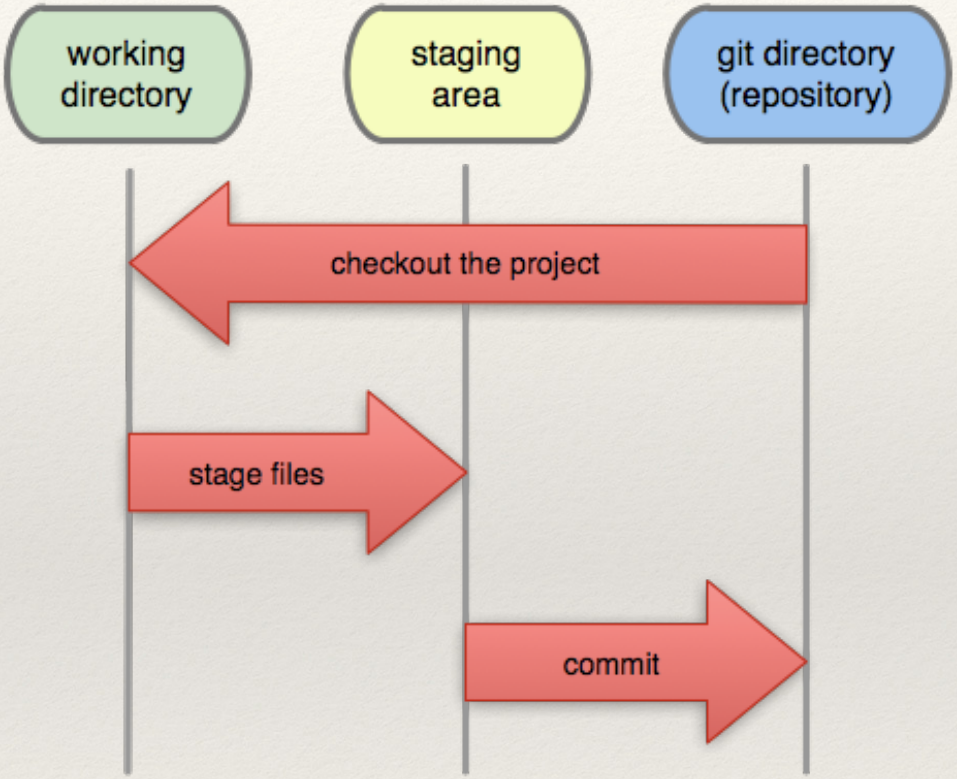
- Use for keeping track of
  - programs, Rscripts..RData files
  - data sets
  - Rmarkdown files (we will cover this later)

Git Basics

- use **git:commit** to enter your project to start
- make a change to your work (Rscript for example)
- 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

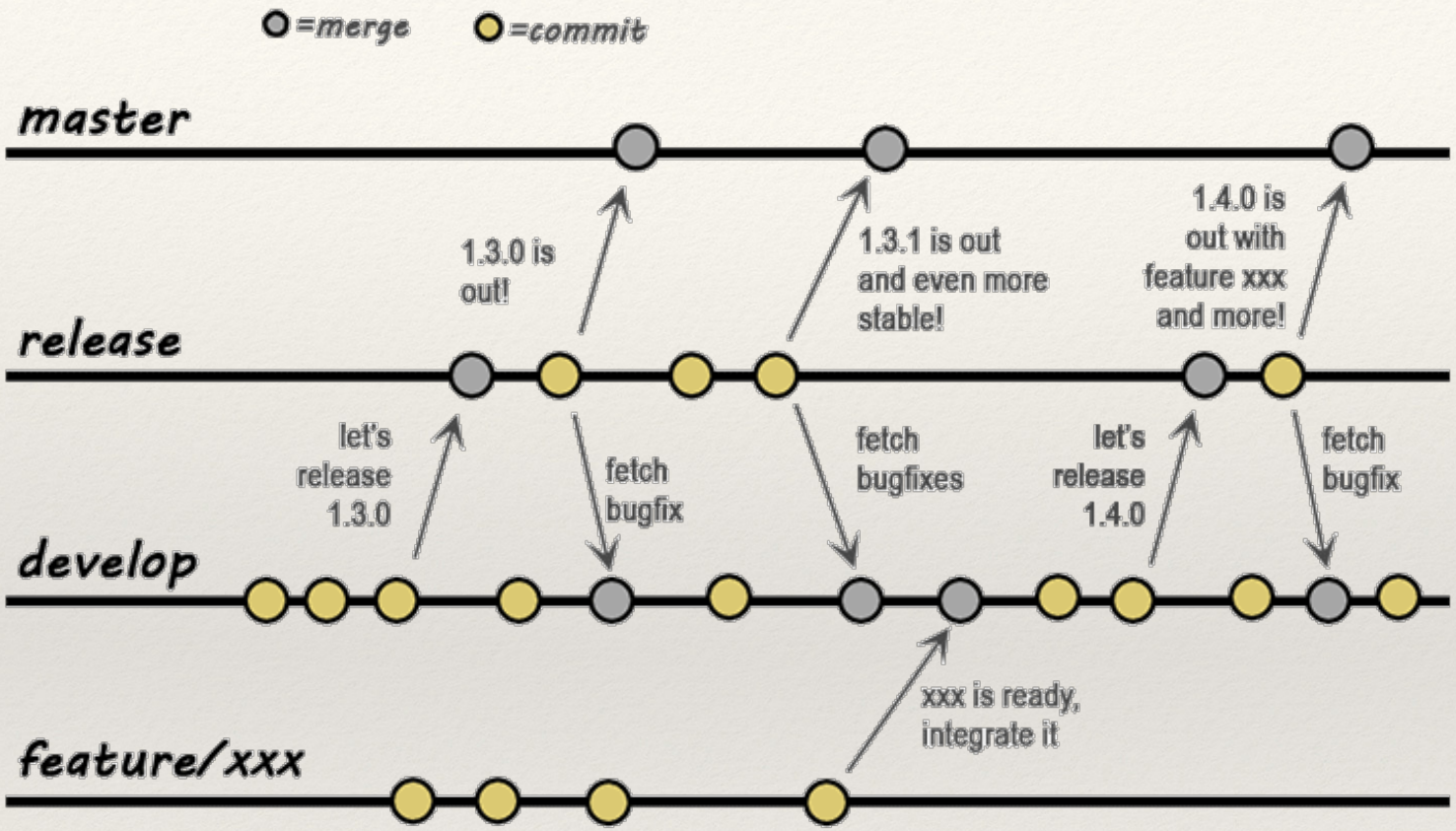
Local Operations



Git Basics (in Rstudio)

- Got to an example - a few things to notice
  - you must save the file first before you can stage it
  - history (under git) will tell you what has changed with each commit
    - HEAD tells you where you are now (HEAD of the tree)
  - as you change a file its status is shown in the Git window
    - M - means you've changed your file
    - A - means you've added a file to the project
  - when you commit make sure your message is a useful one

Git and Branching



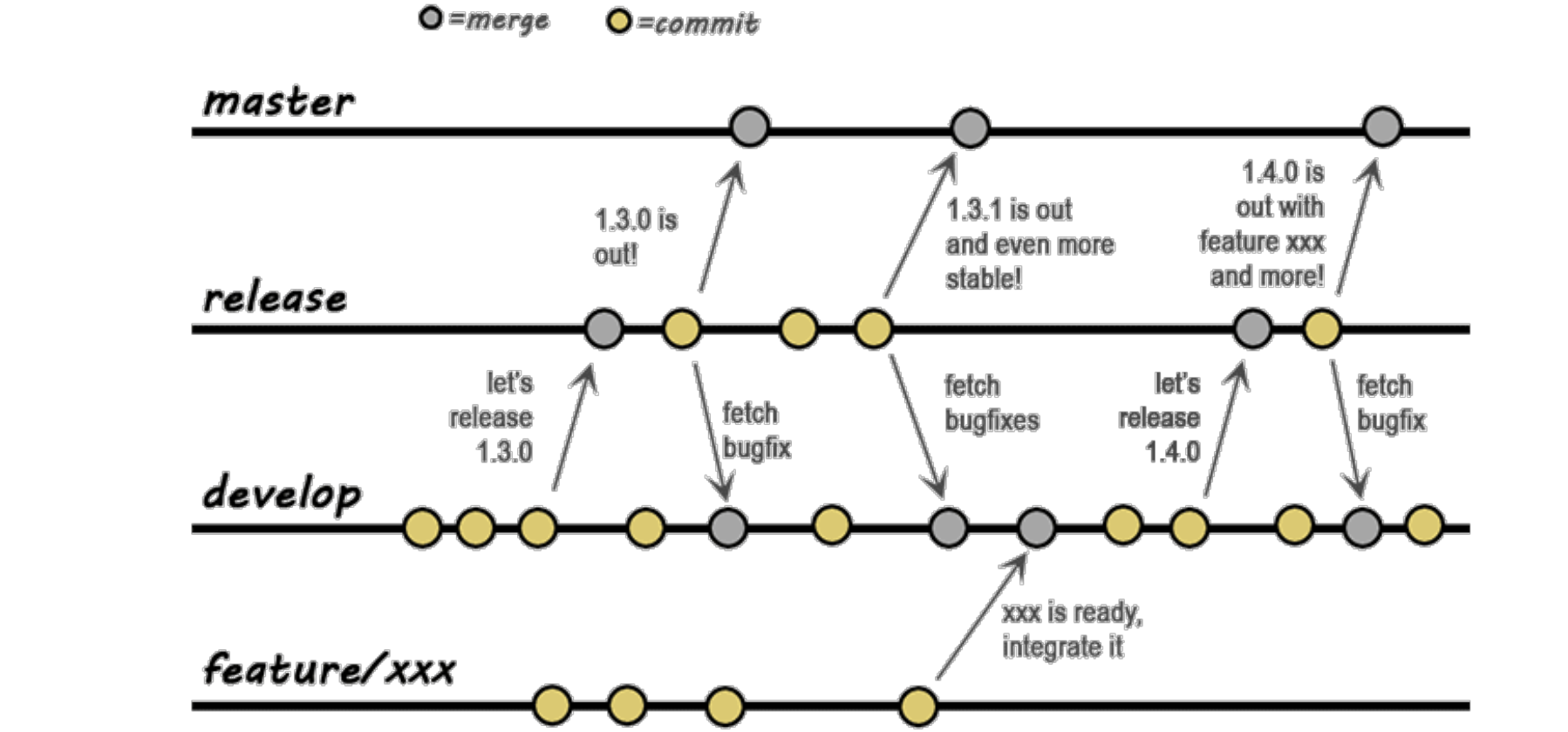
Branch, develop, release are commonly used names of branches
Could be Sara, Experimental, or….

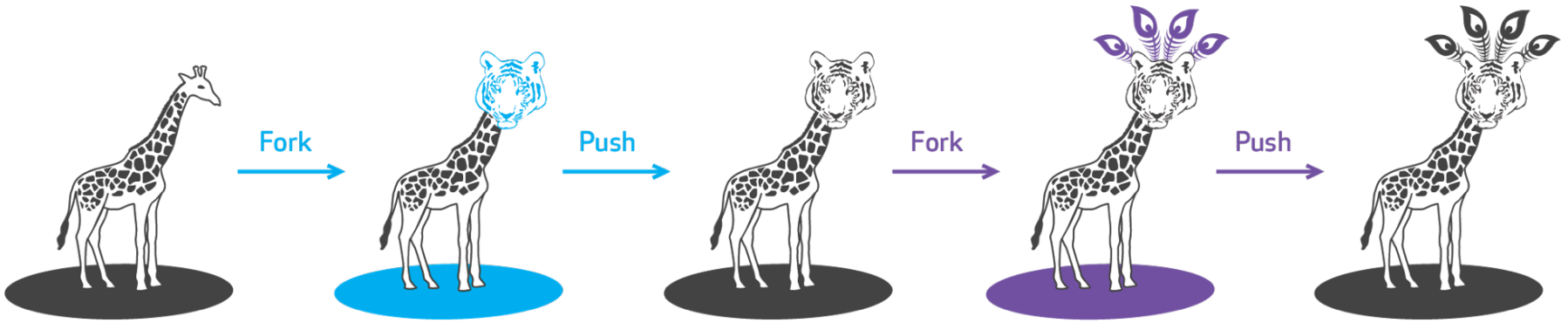
Git and Branching

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

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

Git and Branching

- Work on an example
  - notice how you can see the source of the new branch
  - two reasons to branch
    - a part of your project that might look different (for a client) "client" and never be merged back to master
    - a way for people to work on new ideas/development and then later bring it in to the main project "*naominew*"
      - eventually merged back to master
      - useful to follow the following
        - git checkout *naominew*
        - git merge *master* #bring any changes that have happened to master into your development work
        - make any needed fixes
        - git checkout *master*
        - git merge *naominew* #bring your new development into the main project

Git and Rstudio

- 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 Shell 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 Shell 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):

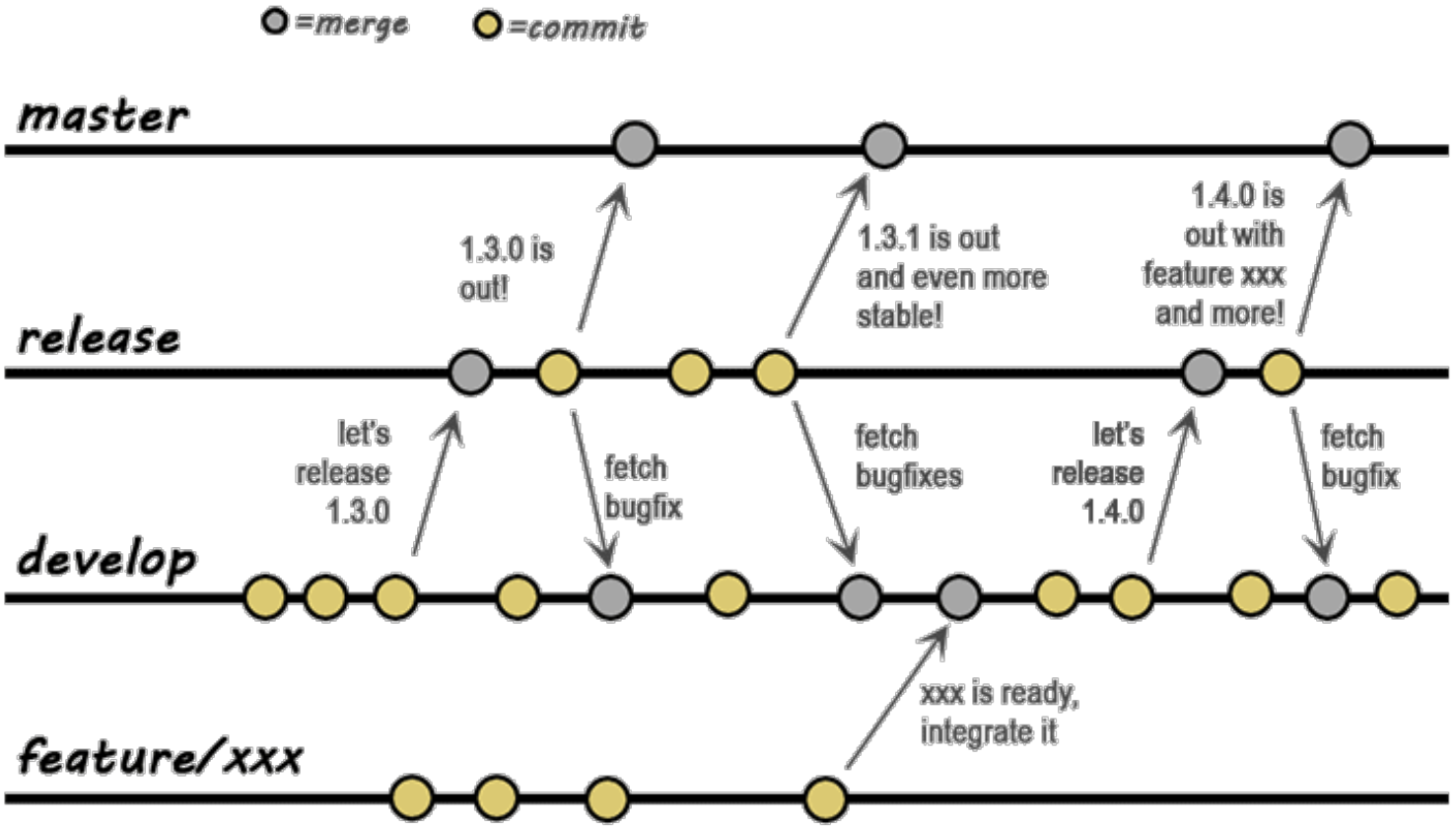
git rm

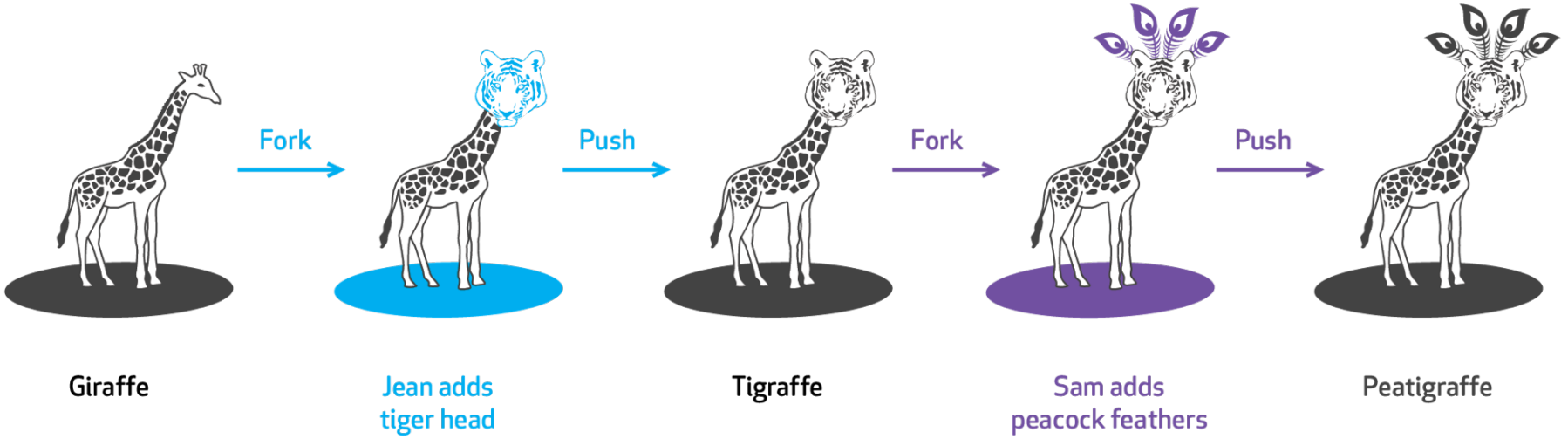
Check the status or your repository – this will list what files have been modified since the last commit, and list any files not currently under version control (i.e. new files you have created:

git status

If you "muck" things up and want to go back

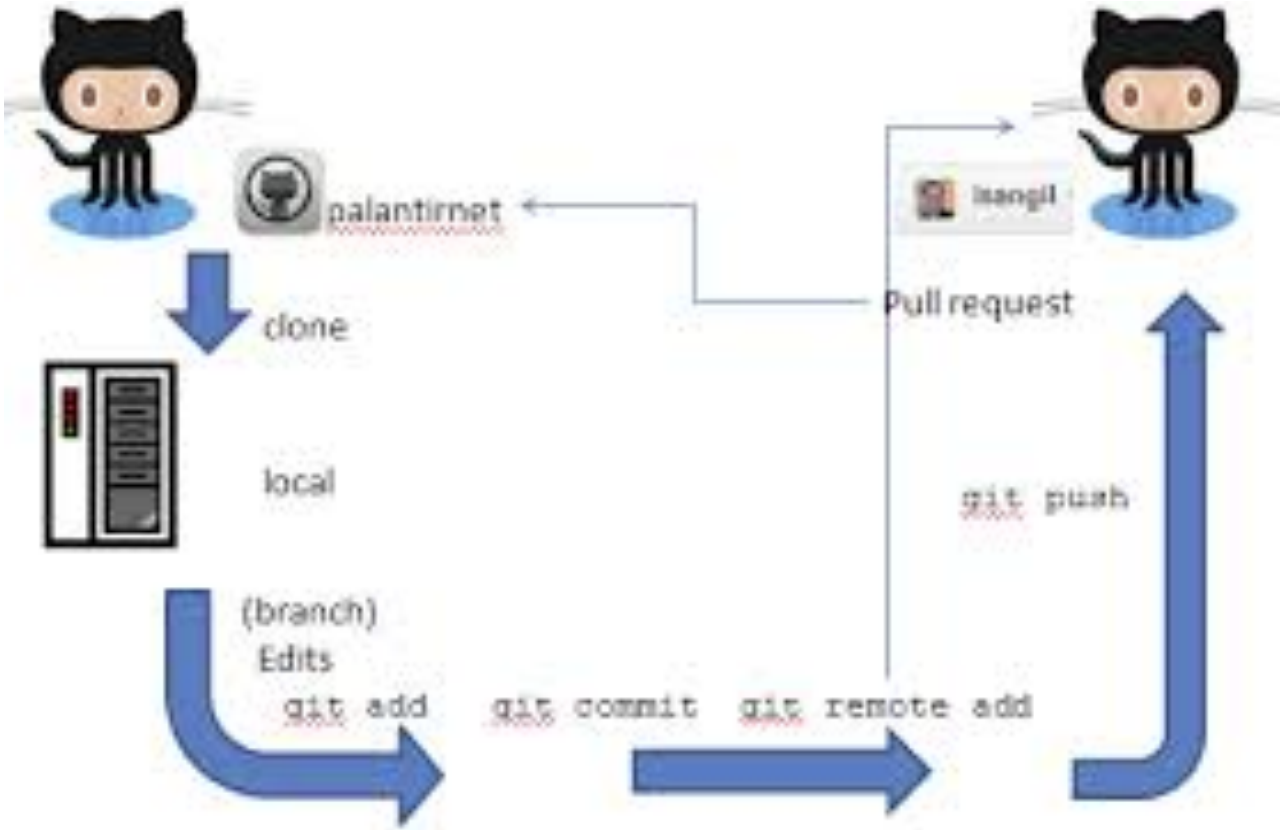
- before you commit: git revert (in Rstudio) or
- after you commit
  - find where you want to go back: in shell
    - git log
    - git log —reverse
    - notice the number
    - git reset #*commit* # leaves changes as "staged" but not 'committed'
    - git reset #*commit* —hard #gets rid of all changes

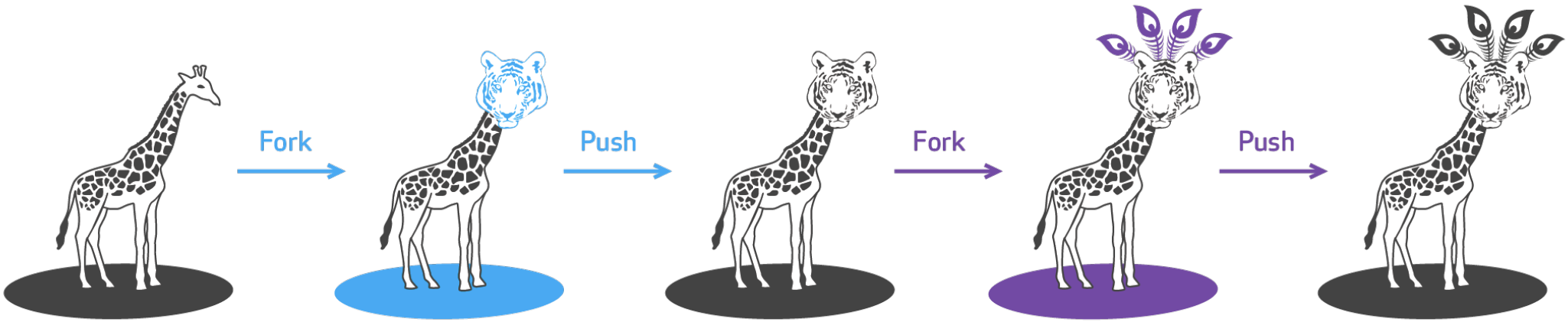

=merge
=commit
master
release
develop
feature/xxx
1.3.0 is out!
1.3.1 is out and even more stable!
1.4.0 is out with feature xxx and more!
let's release 1.3.0
fetch bugfix
fetch bugfixes
let's release 1.4.0
fetch bugfix
xxx is ready, integrate it


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





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

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/
http://r-pkgs.had.co.nz/git.html

Assignment

- With your group
- Create a repository
  - signup for Github (all group members)
  - https://help.github.com/articles/create-a-repo/
    - just one for the group
  - together: generate a project in Rstudio to read, analyze, plot some kind of data that you find interesting
  - push this to the Github repository
  - individual: link to this repository
    - make some changes to the Master branch - upload to github
    - make your own branch - upload to github
    - merge your branch with master - and upload to github
    - make a changes to someone else's branch - upload to github
  - Turn in the link to your repository and a brief description (3-4 sentences) as an Rmarkdown file*
  - *we will explain this shortly