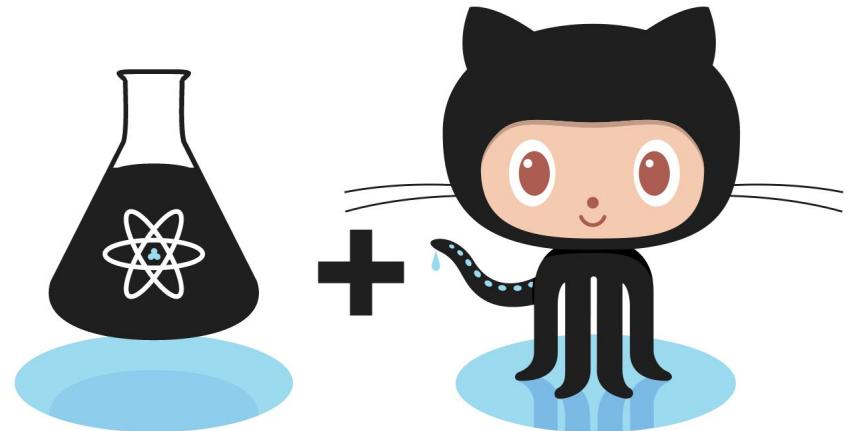


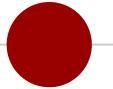
version control for
scientists



Why ‘for scientists’?

- Originally a tool for software developers
- Used (slightly) differently in science





Contents/goals

- **What** is version control?
- **Why** version control?
- **Git** basics/terminology
- Tutorial 1: initialize git repo + connect to Github
- Tutorial 2: collaborating



What is version control?

- A way to manage changes to documents
 - Meant for **plain text files**, not data or “rich text”
(e.g., Word docx)

Rich text can contain formatting and style information for text.

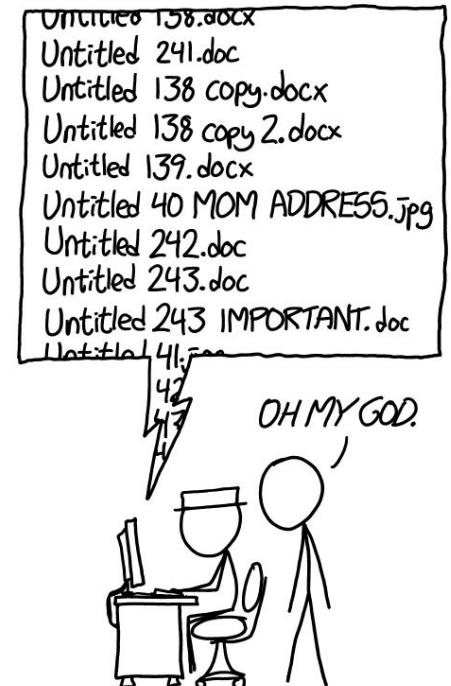
Formatting such as **bold**, **color**, **size**, *italics* and more

Plain text can contain just text, line breaks, and spacing.



What is version control?

- A way to manage changes to documents
- Update versions & revert back to old versions
 - All using the same file, e.g., analysis.m
 - No more analysis_final_version2_really_final_with_plots_nocomments_updated4.m





Why version control?

- Avoid versions across files!
- “Relatively” easy to revert to old versions
- Great for collaboration
- A *must* for software development
- Nice way to make your research more transparent, reproducible, and replicable



Ways to use it in science

- Storage for your code/tools (cf. Dropbox)
 - Might be useful for others!

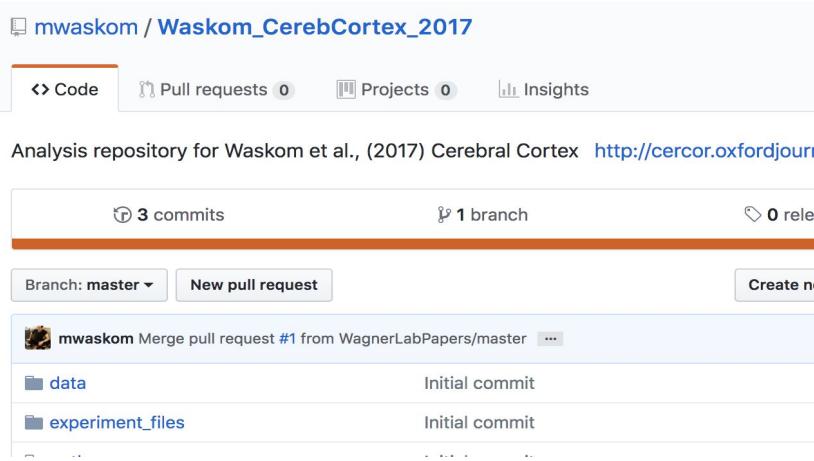
The screenshot shows a GitHub repository page for 'anne-urai / Tools'. The repository has 121 commits, 1 branch, 0 releases, 1 contributor, and is licensed under BSD-2-Clause. The commit history lists several contributions, including patches for CONVNFFT, CircStat2012a, Colormaps, and SDT, with the latest commit being 71c30c1 on 20 Dec 2017.

Commit	Description	Date
CONVNFFT	blinkinterp	2 years ago
CircStat2012a	updated convnfft	2 years ago
Colormaps	bugfix	8 months ago
SDT	hatch	3 months ago



Ways to use it in science

- Publish code along with article
 - Transparency, reproducibility, replicability

A screenshot of a GitHub repository page. The repository name is "mwaskom / Waskom_CerebCortex_2017". The main navigation bar shows "Code" (selected), "Pull requests 0", "Projects 0", and "Insights". Below the bar, it says "Analysis repository for Waskom et al., (2017) Cerebral Cortex <http://cercor.oxfordjournals.org>". It shows 3 commits, 1 branch, and 0 releases. A "Create new" button is visible. The commit history includes a merge pull request from WagnerLabPapers/master. Two files are listed: "data" and "experiment_files", both with initial commits.

Analyses for Waskom et al. (In Press) Cereb Cortex

This repository contains analysis code for the following paper:

Waskom, M.L., Frank, M.C., Wagner, A.D. (In Press). [Adaptive engagement of cognitive control in context-dependent decision-making](#). *Cerebral Cortex*.

The high-level code is contained within several IPython notebooks that performed the analyses and generated all figures in the manuscript. This code makes use of a local library of experiment-specific code and several other libraries that are freely available elsewhere.

Preprocessing

The analysis notebooks assume that most of the heavy processing of the imaging data have already been performed. This can be accomplished in two major steps. First, Freesurfer was used to process the anatomical images and build models of the cortical surface for each subject. Specifically, the following command was used:

```
recon-all -s $subject -all -3T
```

Second, functional timeseries images were processed using [lyman](#). The command lines to reproduce those analyses can be found in the [run_wholebrain.sh](#) script.



Ways to use it in science

- Storage for lab's code
 - Common tools (code)
 - Publications
 - Guidelines

The screenshot shows a GitHub organization profile for 'Knapen Lab'. At the top, there is a blue logo of a winged creature. To the right of the logo, the name 'Knapen Lab' is displayed in large black text. Below the name, the text 'Knapen lab @ Vrije Universiteit Cognitive Psychology' is shown. Underneath this, there are location and contact details: 'Amsterdam', a link to 'http://tknapen.net', and an email address 't.knapen@vu.nl'. Below the header, there are four tabs: 'Repositories 14' (which is highlighted in orange), 'People 15', 'Teams 2', and 'Projects 0'. A search bar labeled 'Search repositories...' is located below the tabs.

[docker_images](#) Private

repo for docker scripts and images in the lab

● Shell Updated 9 hours ago

[response_fytter](#)

fit response shapes from signal time-courses

● Jupyter Notebook MIT Updated 28 days ago

[hires_ODC_7T](#)

Analysis of hires fMRI project

● Jupyter Notebook Updated on 6 Dec 2017



Ways to use it in science

- Storage for lab's code

- Common tools (code)

- Publications

- Guidelines

The screenshot shows a GitHub repository page for 'foraker / lab-manual'. The repository has 7 commits, 1 branch, 0 releases, and 1 contributor. The latest commit was made by jeffcole on June 16, 2016. The repository contains files named process, tools, and README.md. The README.md file is described as 'The Lab Manual'.

foraker / lab-manual

Code Issues 0 Pull requests 0 Projects 0 Insights

Watch 9 Star 0 Fork 1

A manual for development at Foraker Labs

7 commits 1 branch 0 releases 1 contributor

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

jeffcole Merge pull request #4 from foraker/realistic-data Latest commit 3229bcc on 16 Jun 2016

process Add link to 'unapproved-pull-requests' 2 years ago

tools Add and update JS, OOD, and RoR resources 2 years ago

README.md Add "Test With Realistic Data" 2 years ago

README.md

The Lab Manual

The Lab Manual is a guide to development at Foraker Labs. It's where we document our strongly held opinions, our preferred tools, and how we go about particular tasks. It's also where we collect all of the resources that we believe to be essential to doing good work as a developer.

Git(hub) basics





Git vs. Github

- Git is a version control **system** (software)
- Github is a place to **host** your projects that use git

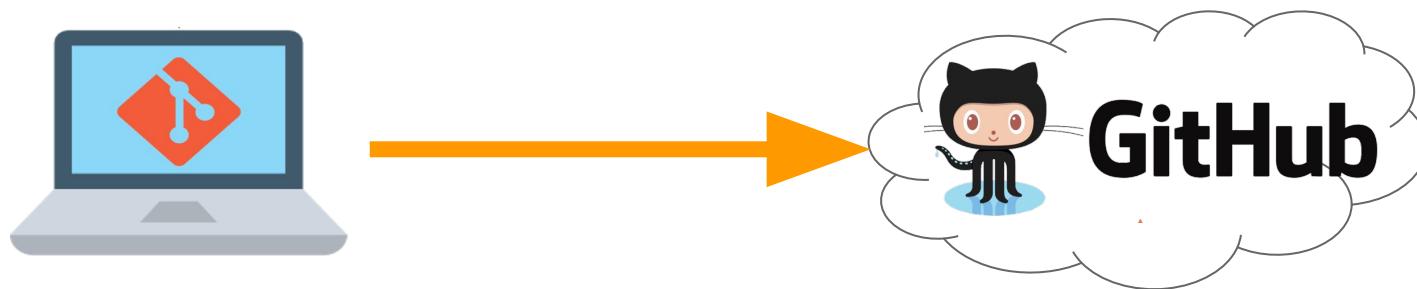




Git vs. Github

In other words ...

- Git is software (used **locally**), while Github is
“hosting service” (a “remote” service)
 - Github ≈ Dropbox (for code)





Git CLI

- Git is **software**
 - Nerd fun fact: developed by Linus Torvalds (creator of Linux!)
- Here, explained from its **command line interface**

```
lukas@Lukass-MacBook-Pro: ~
$ git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]
```



Git CLI

- Git is **software**
 - Nerd fun fact: developed by Linus Torvalds (creator of Linux!)
- Here, explained from its **command line interface**
- (You also have a Github GUI!)

The screenshot shows a GitHub commit interface. At the top, it displays the repository path 'Repositories > Portal > Commit' and the author 'Jeanno' with the date '7 June, 2013 3:04 AM'. Below this is a commit message: 'Add login page signup section style'. On the right, the commit hash '82a449e728156afe3bec68542b6e26a48e748d5b' is shown. The main area shows a diff for the file 'css/user/login.css'. The diff highlights changes in line 14, where the width of an element is being modified from 300px to 360px. The interface includes standard GitHub commit controls like 'Revert commit' and 'Roll back to this commit'.

```
diff --git a/css/user/login.css b/css/user/login.css
--- a/css/user/login.css
+++ b/css/user/login.css
@@ -11,15 +11,15 @@
11 11 border-radius: 5px;
12 12 margin-left: auto;
13 13 margin-right: auto;
14 -width: 300px;
14 +width: 360px;
```



Git repositories

- Git works with “**repositories**”
 - A hidden folder called “.git”



```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ ls -a
.
..
.git
example_file.txt
```



Git repositories

- Git works with “**repositories**”
- The folder containing the “.git” directory is called the **working directory**

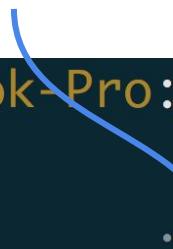
```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ ls -a
.
..
.git
example_file.txt
```



Git repositories

- Git works with “**repositories**”
- The folder containing the “.git” directory is called the **working directory**
 - Git will keep track of **any file** in the working directory

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ ls -a
.
..
.git
example_file.txt
```





Ready to go?

- Git installed?
- Do you have a Github account?



GitHub

Tutorial 1: Git basics





Git repositories

- To create a .git directory (and thus create a “repo”) ...
 - Navigate to the folder you want to place under git control (best to create a new, empty one)
 - Type: `git init`

```
lukas@Lukass-MacBook-Pro: ~
$ mkdir git_tutorial
lukas@Lukass-MacBook-Pro: ~
$ cd git_tutorial/
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git init
Initialized empty Git repository in /Users/lukas/git_tutorial/.git/
```



Git repositories

- Now, we can use other git commands
- The most important one: git status

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
```

```
$ git status  
On branch master
```

```
No commits yet
```

```
nothing to commit (create/copy files and use "git add" to track)
```



Git repositories

- Now, add a file (e.g., a new MATLAB file, text-file, markdown file ...)

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ cat normalize.py
import numpy as np

def demean(arr):
    """Demeans an array (i.e., subtracts the mean
from each value in the array)."""

    arr_demeaned = arr - np.mean(arr)
    return arr_demeaned
```



Git repositories

- Now, add a file (e.g., a new MATLAB file, text-file, markdown file ...)
- Run git status again!

```
$ git status
On branch master

No commits yet

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

    normalize.py

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



Git commits

- Git's major functionality is to create “checkpoints”/“snapshots”, which are called **commits** in git terminology
- These commits reflect “versions” of your code



Git add & git commit

- Git tells us to *first* add files before committing them

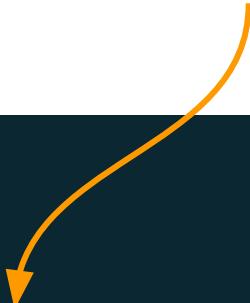
```
$ git status
On branch master

No commits yet

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

    normalize.py

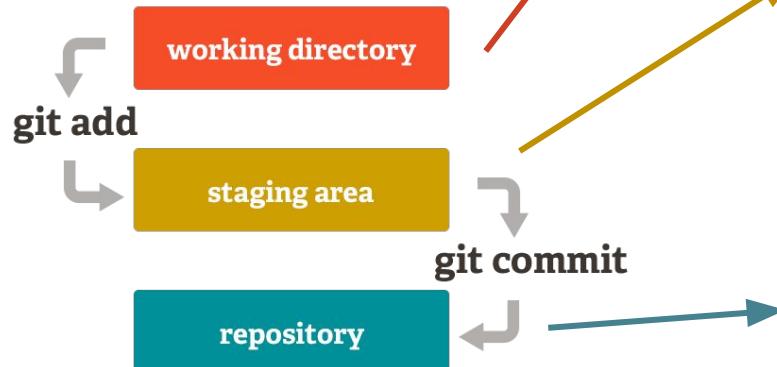
nothing added to commit but untracked files present (use "git add" to track)
```





Git commits

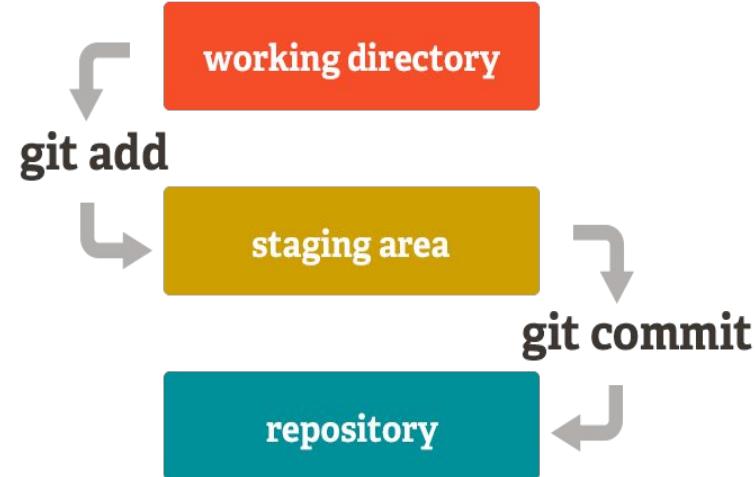
Before committing a file (or files), you need to add them to the “**staging area**”





Git commits

- Importantly, one commit may contain *multiple* files!
- A commit should reflect a coherent “change”
- For example websites: an update is usually reflected as a change in two files (.html + .css)



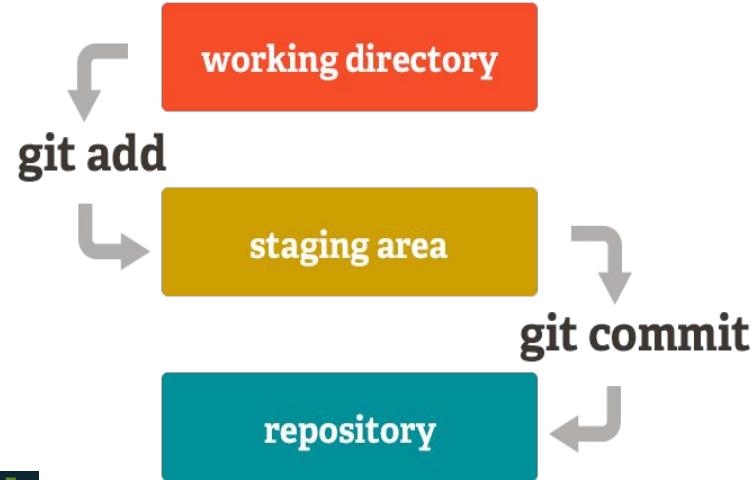


Git commits

- Often, though, a commit contains one file
- Let's try it! We'll add normalize.py (or your own file) to the staging area:

```
git add {your_file}
```

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial  
$ git add normalize.py
```

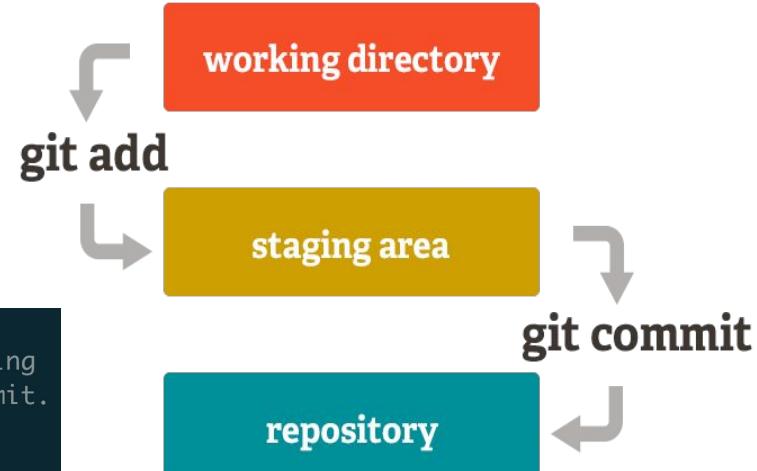




Git commits

- Now, to *commit* your file(s), type: `git commit`
- This will open your system's default editor to add a "commit message"

```
Initial commit
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       modified:   normalize.py
#
```

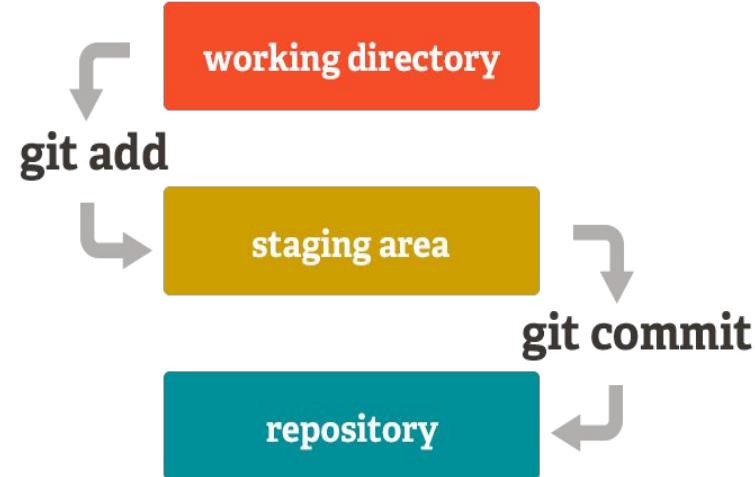




Git commits

- Use `git commit -m "Your commit message"` as a shortcut

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git commit -m "Initial commit"
[master 8fd4baf] Initial commit
 1 file changed, 9 insertions(+)
 create mode 100644 normalize.py
```





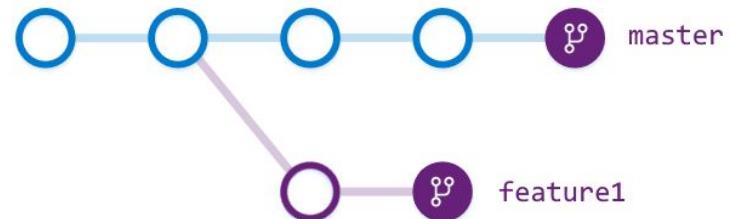
Git branches

- Suppose you want to **change** something about your code ...
 - E.g. new functionality/feature
 - Complete rewrite/restructure (“refactor”)
- ... but you want to keep a **working version** of the original file
 - For example, because others depend on it



Git branches

- Git supports this by “branches”
 - “Copies” of the original branch (the **master** branch)
 - You can switch ‘versions’ by switching branches





Git branches

- Let's check this out. Type `git branch` to get the name of the current branch you're on

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git branch
* master
```



Git branches

- To create a new branch, type:

```
git branch {name_of_new_branch}
```

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git branch new_branch
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git branch
* master
  new_branch
```



Git branches

- Importantly, you're not actually switched to the new branch after calling

```
git branch {name}
```

- To switch to (or to “check out”) your new branch, call:

```
git checkout {name}
```

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git checkout new_branch
Switched to branch 'new_branch'
```



Git branches

- A shortcut to create and immediately check out a new branch, call:

```
git checkout -b {new_branch_name}
```

```
Lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git checkout -b new_branch
Switched to a new branch 'new_branch'
```



Git branches

- Example: I want to extend my demean function to operate on 2D arrays
- I'll create a branch named vectorize
- (But feel free to use your own example!)

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git checkout -b vectorize
Switched to a new branch 'vectorize'
```



Git branches

- After implementing this feature, the file looks like this:

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ cat normalize.py
import numpy as np

def demean(arr, dim):
    """Demeans an array (i.e., subtracts the mean
    from each value) for across rows (dim=0)
    or columns (dim=1)."""

    arr_demeaned = arr - np.mean(arr, axis=dim, keepdims=True)
    return arr_demeaned
```



Git branches

- Now, let's commit this change

```
git add normalize.py
```

```
git commit normalize.py -m "Extend to 2D arrays"
```

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git add normalize.py
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git commit normalize.py -m "Extend to 2D arrays"
[vectorize 37a6b80] Extend to 2D arrays
 1 file changed, 4 insertions(+), 3 deletions(-)
```



Git diff

- Want to know exactly how your files differ across branches?

Use git diff!

```
git diff <ref branch> <current branch>
```

```
$ git diff master vectorize
diff --git a/normalize.py b/normalize.py
index 8c4c335..2c8df97 100644
--- a/normalize.py
+++ b/normalize.py
@@ -1,9 +1,10 @@
 import numpy as np

-def demean(arr):
+def demean(arr, dim):
    """Demeans an array (i.e., subtracts the mean
- from each value in the array)."""
+ from each value) for across rows (dim=0)
+ or columns (dim=1)."""

- arr_demeaned = arr - np.mean(arr)
+ arr_demeaned = arr - np.mean(arr, axis=dim, keepdims=True)
return arr_demeaned
```



Git branches

- ➊ Alright, let's switch back to our master branch:

```
git checkout master
```

- ➋ And verify that it contains the original file!



Git branches

Check out the file on
your file system:

the file itself is really
different!

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git checkout master
Switched to branch 'master'
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ cat normalize.py
import numpy as np

def demean(arr):
    """Demeans an array (i.e., subtracts the mean
    from each value in the array)."""

    arr_demeaned = arr - np.mean(arr)
    return arr_demeaned
```



Git merge

- Suppose we're happy with our new feature on our vectorize branch - how do we get it back in our master branch?
- git merge!
 - Smart merging of the same file across branches

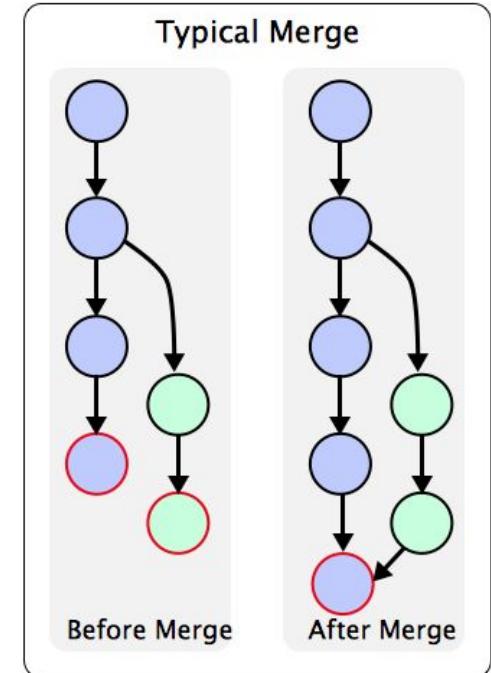


Git merge

- To merge a branch:

```
git merge {branch_to_merge_in}
```

- Importantly, you do this *from the reference branch* (usually master)



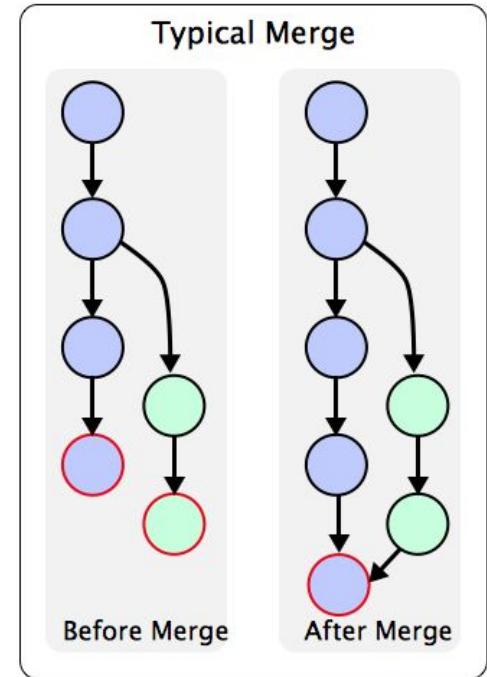


Git merge

- Example:

```
git checkout master  
git merge vectorize
```

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial  
$ git checkout master  
Switched to branch 'master'  
lukas@Lukass-MacBook-Pro: ~/git_tutorial  
$ git merge vectorize  
Updating 8fd4baf..37a6b80  
Fast-forward  
 normalize.py | 7 +----  
 1 file changed, 4 insertions(+), 3 deletions(-)
```





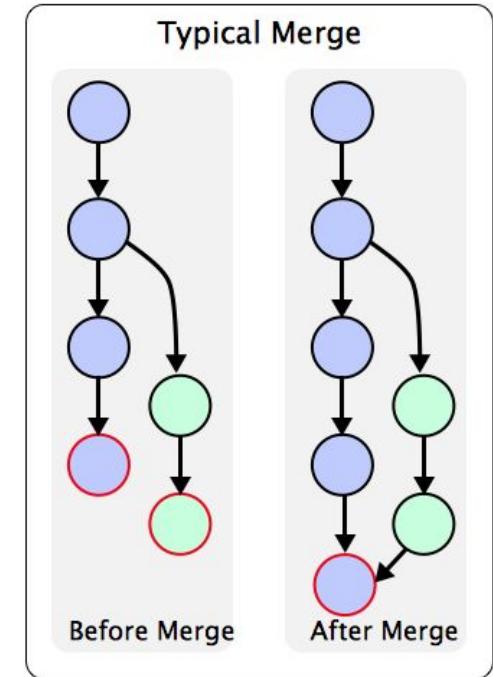
Git merge

- ➊ It worked!

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git branch
* master
  vectorize
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ cat normalize.py
import numpy as np

def demean(arr, dim):
    """Demeans an array (i.e., subtracts the mean
    from each value) for across rows (dim=0)
    or columns (dim=1)."""

    arr_demeaned = arr - np.mean(arr, axis=dim, keepdims=True)
    return arr_demeaned
```





Adding a remote

- So far, we've only worked **locally** with git
- Usually, we want to “sync” it to a **remote**
- A “remote” can be any (web)server/host!
 - I'll show how to use **Github**, but other host services exist (Gitlab, Bitbucket)
 - Even an (SSH-accessible) **server** would work, like the TUX-servers at the UvA, or your home PC



Adding a remote

- ◉ Note that Github is a commercial service!
 - But free for academic use (i.e., if you have a @uva or any other academic email address)



Github

- ➊ Apply for a free academic account!
- ➋ [Link](#)

Applying for a free Developer plan for your personal account

1 Go to GitHub's [Education](#) page and click **Request a discount**.



2 Select **Researcher**.



3 To apply this discount to your personal account, select **Individual account** and click **Next**.



4 Type your name.



5 Verify your academic status.

› If you have a school-issued email address, use the drop-down menu and click the desired email address.

› If you don't have a school-issued email address, use the drop-down menu and click **I don't have a school-issued email address**. Next, upload an image of your school or faculty ID, employment verification letter, academic transcript, or other proof of affiliation that shows at least one date to prove your current academic status.



6 Enter the school name.



7 Describe how you plan to use GitHub in your research group.



8 Verify your application details, then click **Submit Request**.

If your application is approved, you'll receive a confirmation email. Applications are usually processed within a few days, but it may take longer during peak times, such as during the start of a new semester.





Adding a remote

- Note that Github is a commercial service!
 - But free for academic use (i.e., if you have a @uva or any other academic email address)
- I use it because it offers all kinds of cool features/integrations
 - Automatic testing and documentation generation
 - Create and host websites with a single click



Adding a remote

MVPA of fMRI data in Python

ICON 2017 workshop

[View on GitHub](#) [Download material \(.zip\)](#) [Main page](#)



Hi there!

This is the website for the **multivoxel pattern analysis (MVPA) of fMRI data in Python** workshop which was originally given at [ICON 2017](#). The materials and data will remain online so that everyone who's interested can follow the workshop. Also, all the material is *fully open-source*, so feel free the re-use and modify this workshop's materials to fit your own educational activities! If you find mistakes or have suggestions for improvements of this workshop, please let me know by email (see [Github](#)) or, even better, submit a "Pull Request" (PR) to this workshop's [Github repository](#) - this is greatly appreciated! That said, on this website, you'll find some general info about the workshop, how to prepare for it (in terms of Python prerequisites and environment), and the slides/notebooks/data that will be used during the workshop.

Also, check out the slides from the introductory lecture below:

lukassnoek / ICON2017

Code Issues Pull requests Projects Wiki Insights Settings

Repository for the ICON 2017 hackathon 'multivoxel pattern analysis (MVPA) of fMRI data in Python' <http://lukassnoek.com/ICON2017>

fMRI MVPA Python scikit-learn machine-learning Manage topics

102 commits 1 branch 0 releases 3 contributors MIT

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

lukassnoek Add files via upload Latest commit [89afab](#) on Nov 27, 2017

File	Description	Age
_layouts	Update default.html	8 months ago
code	Add python tutorial	8 months ago
lecture_slides	Rename zipfile and add lecture slides	8 months ago
presentation	Add files via upload	3 months ago
tutorial	Fix minor bugs and win/unix stuff in paths	7 months ago

Hi there!

This is the website for the **multivoxel pattern analysis (MVPA) of fMRI data in Python** workshop which was originally given at [ICON 2017](#). The materials and data will remain online so that everyone who's interested can follow the workshop. Also, all the material is *fully open-source*, so feel free the re-use and modify this workshop's materials to fit your own educational activities! If you find mistakes or have suggestions for improvements of this workshop, please let me know by email (see [Github](#)) or, even better, submit a "Pull Request" (PR) to this workshop's [Github repository](#) - this is greatly appreciated! That said, on this website, you'll find some general info about the workshop, how to prepare for it (in terms of Python prerequisites and environment), and the slides/notebooks/data that will be used during the workshop.

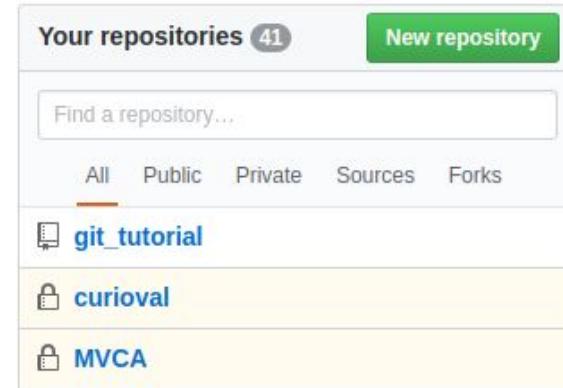
Also, check out the slides from the introductory lecture below:

```
<style> .responsive-wrap iframe{ max-width: 100%; } </style>
<iframe src="https://docs.google.com/presentation/d/1bgK1sv-VmnVAR_HgTiWsU0Qkjz6oO1zQ5ly4laPkw/embed?
start=false&loop=false&delayms=3000" frameborder="0" width="900" height="569" allowfullscreen="true"
mozallowfullscreen="true" webkitallowfullscreen="true"></frame>
```



Adding a remote

- To add a “remote” to our local git repo, we need to create one first!
- Go to Github and click “New repository”





Adding a remote

A .gitignore is a “config file” in which you can indicate which files git should ignore (e.g. *.mat files)

Create a new repository
A repository contains all the files for your project, including the revision history.

Owner	Repository name
lukassnoek ▾	/ your_repo_name

Great repository names are short and memorable. Need inspiration? How about [psychic-succotash](#).

Description (optional)
This is my first (?) remote repository!

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: Python ▾ | Add a license: MIT License ▾

Create repository

A license indicates if (and to what extent) people can re(use) and modify your code



Adding a remote

- Now, we have a **local repository** (on your laptop) and a **remote repository** (on Github)
- We link them using the command:

```
git remote add
```

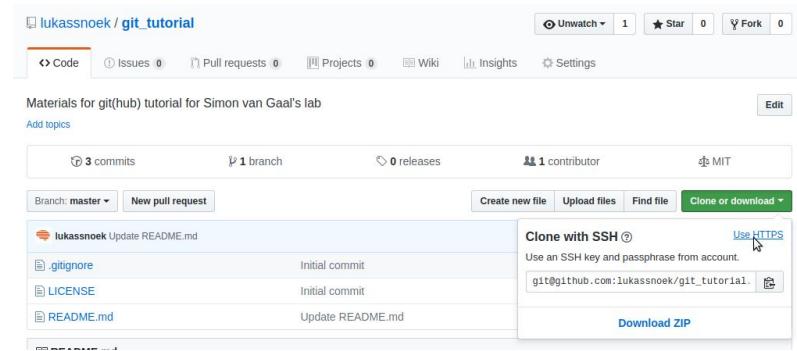
```
usage: git remote add [<options>] <name> <url>
      -f, --fetch           fetch the remote branches
      --tags                import all tags and associated objects when fetching
                            or do not fetch any tag at all (--no-tags)
      -t, --track <branch> branch(es) to track
      -m, --master <branch>
                            master branch
      --mirror[=<push|fetch>]
                            set up remote as a mirror to push to or fetch from
```

```
usage: git remote add [<options>] <name> <url>
          -f, --fetch           fetch the remote branches
          --tags                import all tags and associated objects when fetching
                                or do not fetch any tag at all (--no-tags)
```



Adding a remote

- It's common practice to name your remote "origin"
- To get the url of your remote, click the "Clone or download" button
- You can download the material using SSH-authentication or via HTTPS



```
usage: git remote add [<options>] <name> <url>
      -f, --fetch                  fetch the remote branches
      --tags                       import all tags and associated objects when fetching
                                    or do not fetch any tag at all (--no-tags)
```



Adding a remote

- Copy the link (either SSH or HTTPS-based) and call:

Prints out the currently connected remote

```
git remote add origin git@github.com:lukassnoek/git_tutorial.git
```

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git remote add origin git@github.com:lukassnoek/git_tutorial.git
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git remote -v
origin  git@github.com:lukassnoek/git_tutorial.git (fetch)
origin  git@github.com:lukassnoek/git_tutorial.git (push)
```



Syncing a local repo & remote

- Now, we can push our local repository to Github using the command:

```
git push -u [remote name] [local branch name]  
git push -u origin master
```



Syncing a local repo & remote

- But it fails! Why?

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git push -u origin master
To github.com:lukassnoek/git_tutorial.git
 ! [rejected]      master -> master (fetch first)
error: failed to push some refs to 'git@github.com:lukassnoek/git_tutorial.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```



Syncing a local repo & remote

- If the remote contains new/modified files, you need to pull the remote first:
 - `git pull [remote name] [remote branch name]`
`--allow-unrelated-histories`
 - `git pull origin master --allow-unrelated-histories`
- The `git pull` will trigger a merge between the remote files and the local files



Syncing a local repo & remote

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial  
$ git pull origin master --allow-unrelated-histories
```

```
Merge branch 'master' of github.com:lukassnoek/git_tutorial  
  
# Please enter a commit message to explain why this merge is necessary,  
# especially if it merges an updated upstream into a topic branch.  
# From github.com:lukassnoek/git_tutorial  
#           * branch      master    -> FETCH_HEAD  
# Lines starting with Merge made by the 'recursive' strategy.          je aborts  
# the commit.  
        .gitignore | 101 ++++++  
        LICENSE   |  21 ++++++  
        README.md |   5 +++  
3 files changed, 127 insertions(+)  
create mode 100644 .gitignore  
create mode 100644 LICENSE  
create mode 100644 README.md
```



Syncing a local repo & remote

- Now, we're really ready to push to our remote:

```
git push -u origin master
```

```
lukas@Lukass-MacBook-Pro: ~/git_tutorial
$ git push -u origin master
Counting objects: 14, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (14/14), 1.37 KiB | 704.00 KiB/s, done.
Total 14 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To github.com:lukassnoek/git_tutorial.git
  2d2f11d..61c8186  master -> master
Branch master set up to track remote branch master from origin.
```



Syncing a local repo & remote

- Check out your Github repo! Are your new files/changes there?

Branch: master ▾ New pull request

Create new file Upload files Find file Clone or download ▾

lukassnoek Merge branch 'master' of github.com:lukassnoek/git_tutorial		Latest commit 61c8186 4 minutes ago
.gitignore	Initial commit	7 days ago
LICENSE	Initial commit	7 days ago
README.md	Update README.md	23 hours ago
normalize.py	Extend to 2D arrays	21 minutes ago



Initializing a git repo

- An **alternative** (better) **method** to initializing a git repo:
 - *first* create a remote (Github) repo
 - “Clone” (fancy word for download) your remote repo to your laptop:

```
git clone git@github.com:lukassnoek/git_tutorial.git
```



Initializing a git repo

- Best when you have no local files to start with (e.g., completely new projects)
- Avoids the ‘unrelated histories’ error in `git push`



Questions?

Questions so far?

Tutorial 2: collaborating





Collaborating

- Pair up with someone else!
- Pick one repo to work on together
- The owner of the repo adds the other person to the repo as collaborator
 - Settings → Collaborators

Collaborators	Push access to the repository
This repository doesn't have any collaborators yet. Use the form below to add a collaborator.	
Search by username, full name or email address You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.	
<input type="text"/>	<input type="button" value="Add collaborator"/>



Collaborating

Exercise:

- The collaborator (i.e., not owner) clones the repo to his/her laptop:
`git clone <owner's repo url>`
- Make a new branch (`git branch <new name>`) and switch to branch
- Modify/add stuff; commit changes
- Push branch to remote (`git push -u origin <new name>`)
- Owner pulls new branch (`git pull -u origin <new name>`)
- Owner merges new branch into master (`git merge <new name>`)



Don't forget ...

- Git is *hard*



Katie Cunningham @kcunning · Feb 16

I am an experienced developer.

I am also googling "How to roll back a git commit" for the millionth time.

234

1.9K

9.2K



THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.





Don't forget ...

- Git is *hard*
- But it is *useful*
- Takes a while to get used to
 - But a great skill to have (also outside academia!)
- Best way to learn is by *doing* it
- Good luck!