

Version control management and research collaboration using git and github

An introduction

APSYS group

MCC Berlin

July 11th, 2019

What is git and GitHub?

Why should I use it?

How can I use it?

What is version control management?

Software to keep track of the history and different versions of files within project folders

What is git?

- git is a program for version control
- designed for distributed software development
- created by Linus Torvalds for his work on the Linux kernel

Explain idea of a git repository

What is GitHub?

Explain the idea of a remote repository

Explain github (and providers of remote repositories like gitlab, bitbucket, SourceForge, Launchpad ...)

What is git and GitHub?

Why should I use it?

How can I use it?

Why use version control in research?

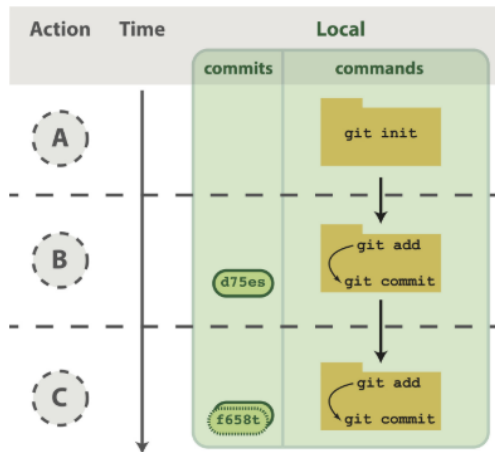
- getting some order in the mess
 - data
 - software code/scripts
 - manuscripts for papers
- sharing your code or
- collaboration and attribution of work

What is git and GitHub?

Why should I use it?

How can I use it?

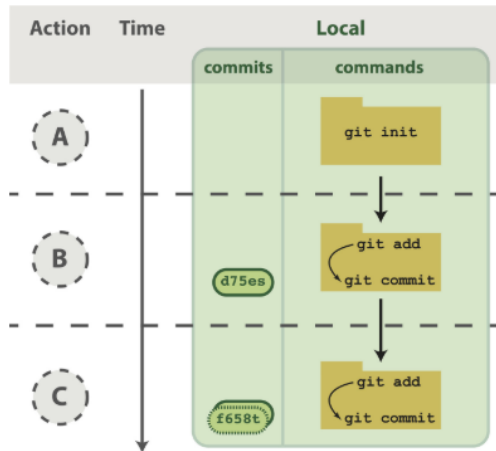
Git Workflow (simplest)



- Keep track of changes in a folder on your computer

Figure: Blischak et al. (2016)

Git Workflow (simplest)



- Keep track of changes in a folder on your computer
- Changes are stored as lines added and removed

Figure: Blischak et al. (2016)

Git Workflow (simplest)

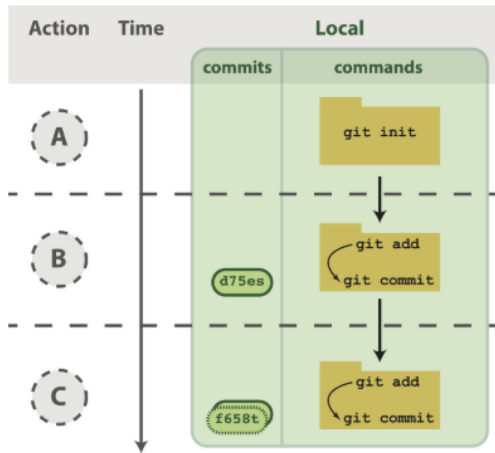
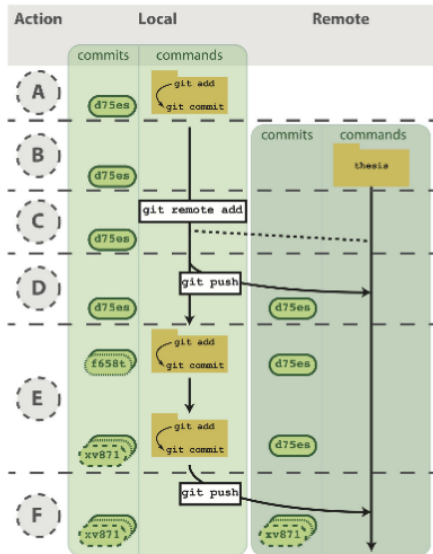


Figure: Blischak et al. (2016)

- Keep track of changes in a folder on your computer
- Changes are stored as lines added and removed
- No need to save multiple versions of the same file; you have recorded all changes and can view or revert these at any time

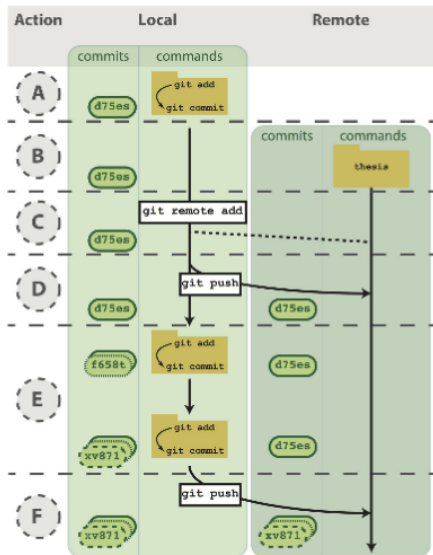
Git + Github Workflow (simplest)

- Attach your repository to a remote version



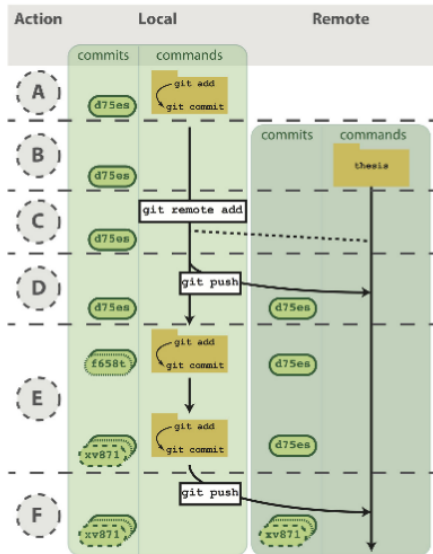
Git + Github Workflow (simplest)

- Attach your repository to a remote version
- If working with collaborators, they also can make a copy (clone) on their machine

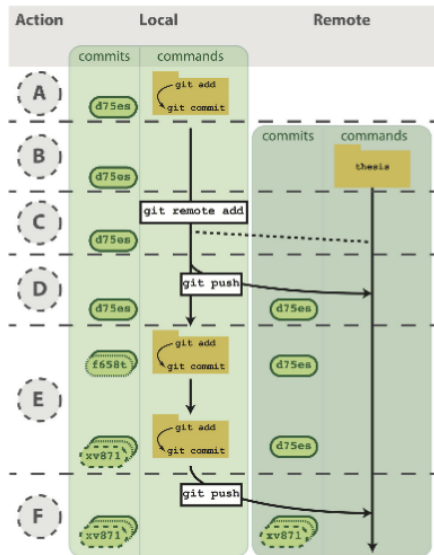


Git + Github Workflow (simplest)

- Attach your repository to a remote version
- If working with collaborators, they also can make a copy (clone) on their machine
- By both using pull, you can keep up to date with each others' changes



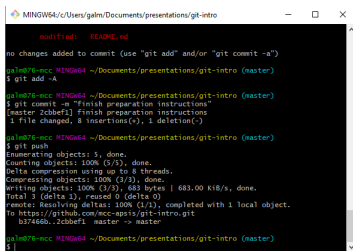
Git + Github Workflow (simplest)



- Attach your repository to a remote version
- If working with collaborators, they also can make a copy (clone) on their machine
- By both using pull, you can keep up to date with each others' changes
- For more complicated workflows, especially where maintaining a working version is critical, check out branching <https://guides.github.com/introduction/flow/>

Command Line

- Easy to document/explain



```
mingw64/c/Users/galm/Documents/presentations/git-intro
modified: README.md

no changes added to commit (use "git add" and/or "git commit -a")
galm@676-mcc MINGW64 ~/Documents/presentations/git-intro (master)
$ git add -A

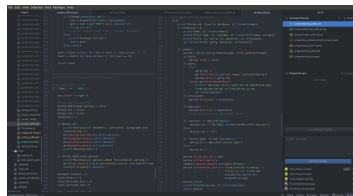
galm@676-mcc MINGW64 ~/Documents/presentations/git-intro (master)
$ git commit -m "Finish preparation instructions"
[master 2c0bef2] Finish preparation instructions
1 file changed, 8 insertions(+), 1 deletion(-)

galm@676-mcc MINGW64 ~/Documents/presentations/git-intro (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 683 bytes | 683.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
to https://github.com/mcc-apsis/git-intro.git
b37466b..2c0bef1 master -> master
galm@676-mcc MINGW64 ~/Documents/presentations/git-intro (master)
$
```

Steeper learning curve, but more flexible and harder to do things unintentionally

GUIs

- Easy to use



Often there are integrations in development environments, e.g. RStudio, Atom

Starting a repository

To start working with a repository, either turn an existing folder into a git repository

```
git init
```

or copy an existing remote repository into a folder on your machine

```
git clone
```

Editing a respository

- Edit files (write some new code or a nice new paragraph)
-

Editing a repository

- Edit files (write some new code or a nice new paragraph)

- Stage changes (tell git about the changes you want record)
 - `git add -A`
 - Or add only certain files using patterns or exact file names

Editing a repository

- Edit files (write some new code or a nice new paragraph)

- Stage changes (tell git about the changes you want record)
 - `git add -A`
 - Or add only certain files using patterns or exact file names

- Commit changes (make a timestamped version of the repository, recording all the changes you have told git about)
 - `git commit -m "made a cool new graph"`
 - It's best if each commit describes a discrete change, and has an interpretable name.

Managing the repository

Where are we?

git status tells us which files have changed and are staged or unstaged:

```
galim@76-mcc MINGW64 ~/Documents/presentations/git-intro (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   slides/presentation.pdf
        modified:   slides/presentation.tex

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   slides/images/git-terminal.png
```

What's changed?

git diff lets us know the difference between the files we could stage, and the staged version of them

Managing the repository

Where are we?

git status tells us which files have changed and are staged or unstaged:

What's changed?

git diff lets us know the difference between the files we could stage, and the staged version of them

```
$ git diff slides/presentation.tex
diff --git a/slides/presentation.tex b/slides/presentation.tex
index 5d65679..3c1a020 100644
--- a/slides/presentation.tex
+++ b/slides/presentation.tex
@@ -274,14 +274,18 @@ or copy an existing repository into a folder

\medskip

- \code{git status}
+ \code{git status} tells us which files have changed and are staged or unstaged:
+
+ \begin{figure}
+   \includegraphics[width=\linewidth]{images/git-status}
+ \end{figure}

- \medskip
+ \medskip

\textbf{What's changed}

- \code{git diff}
+ \code{git diff} lets us know the difference between the files we could stage, and the staged version of them
```

Managing the repository

Where are we?

`git status` tells us which files have changed and are staged or unstaged:

What's changed?

`git diff` lets us know the difference between the files we could stage, and the staged version of them

`git diff` can also tell us about the difference between variously specified versions of files

Navigating different versions

git log shows us a list of all the commits that have been made.

```
galml376-mcc MINGW64 ~/Documents/presentations/git-intro (master)
$ git log
commit 2cbbef100ff738f182676f7bfe420691340a23be (HEAD -> master, origin/master, or
EAD)
Author: Max Callaghan <max.w.callaghan@gmail.com>
Date: Mon Jul 8 15:14:06 2019 +0200

    finish preparation instructions

commit b37466bf9a88bd481887769415375f1e95343f3a
Author: Max Callaghan <max.w.callaghan@gmail.com>
Date: Mon Jul 8 15:03:56 2019 +0200

    preparation instructions in README.md

commit 569f525ab48e1fb922495b204ab1b91d07229b40
Author: Max Callaghan <max.w.callaghan@gmail.com>
Date: Mon Jul 8 13:50:34 2019 +0200

    updated slides
```

git checkout takes you to another commit, or another branch of the repository

git checkout master takes you back to the most recent commit on the master branch

Interacting with remote repositories

If you started a repository on your computer, you can associate it with an *empty* GitHub repository. Follow the instructions on Github:

```
git remote add origin  
https://github.com/mcallaghan/blablabala.git  
  
git push -u origin master
```

Now your local repository exists online, where it's current status and entire history is reflected.

N.B. when dealing with the public domain, don't forget licenses (of your and other people's work).

Interacting with remote repositories

`git pull` downloads all of the changes (commits) that are published in the remote version, and incorporates them (as long as they do not clash with your changes) into your local copy

Merging is normally done automatically, but you may need to choose between different versions if they have both changed the same lines of code

`git push` adds all of your committed changes to the remote version (as long as your version contains all of the changes in the remote - you need to pull first)

You can only push to repositories you have the correct permissions for, but you can make a pull request (a suggestion that your changes be merged into the master) for any repository. This allows changes to be reviewed and tested before incorporation.

Further useful commands and tools

- `.gitignore` - A list of files, or patterns for git to ignore
- Zenodo lets you get a citable DOI for your code/data
- GitHub Pages allows you to quickly turn your repository into a static website
- `Blame` lets you trace, line by line, who contributed which parts of the code, and when

Github also has nice tools for making wikis and tracking issues, among other things.

Questions?

Practice / task

- Make a new project folder, and initialise a git repository with `git init`
- Add a README.md file with some text in it.
- Stage and commit your changes with `git add`, and `git commit`
- Make a new empty repository on Github
<https://github.com/new>
- Follow the instructions to add this remote to your local repository with `git remote add origin [link]` and push your changes `git push -u origin master`
- Find a partner, add each other as collaborators, edit each other's repositories, and push changes.

References

Blischak, J. D., Davenport, E. R., and Wilson, G. (2016). A Quick Introduction to Version Control with Git and GitHub. *PLOS Computational Biology*, 12(1):e1004668.