Version Control with Git

Best Practices Workshop, March 25-26 2019, Hampton VA

Carlos Cruz
Jules Kouatchou
Brent Smith

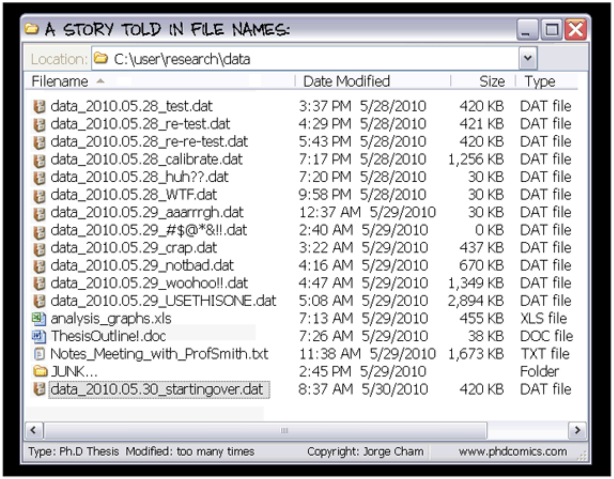
NASA GSFC Code 606/610 (ASTG/GMAO)
Greenbelt, Maryland 20771

Overview

- What is version control (and why do we care) ?
- Centralized vs distributed version control
- Git
    - Basic concepts
    - Usage
    - GitHub
    - Lots of exercises

What is version control?

Many of us constantly create something, save it, change it, then save it again



A STORY TOLD IN FILE NAMES:

Location: C:\user\research\data

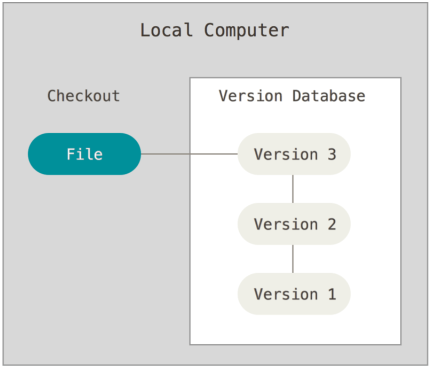
Filename	Date Modified	Size	Type
data_2010.05.28_test.dat	3:37 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_re-test.dat	4:29 PM 5/28/2010	421 KB	DAT file
data_2010.05.28_re-re-test.dat	5:43 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_calibrate.dat	7:17 PM 5/28/2010	1,256 KB	DAT file
data_2010.05.28_huh??.dat	7:20 PM 5/28/2010	30 KB	DAT file
data_2010.05.28_WTF.dat	9:58 PM 5/28/2010	30 KB	DAT file
data_2010.05.29_aaarrrgh.dat	12:37 AM 5/29/2010	30 KB	DAT file
data_2010.05.29_#$@*&!!.dat	2:40 AM 5/29/2010	0 KB	DAT file
data_2010.05.29_crap.dat	3:22 AM 5/29/2010	437 KB	DAT file
data_2010.05.29_notbad.dat	4:16 AM 5/29/2010	670 KB	DAT file
data_2010.05.29_woohoo!!.dat	4:47 AM 5/29/2010	1,349 KB	DAT file
data_2010.05.29_USETHISONE.dat	5:08 AM 5/29/2010	2,894 KB	DAT file
analysis_graphs.xls	7:13 AM 5/29/2010	455 KB	XLS file
ThesisOutline!.doc	7:26 AM 5/29/2010	38 KB	DOC file
Notes_Meeting_with_ProfSmith.txt	11:38 AM 5/29/2010	1,673 KB	TXT file
JUNK...	2:45 PM 5/29/2010		Folder
data_2010.05.30_startingover.dat	8:37 AM 5/30/2010	420 KB	DAT file

Type: Ph.D Thesis Modified: too many times Copyright: Jorge Cham www.phdcomics.com

What is version control?

Version control is a means of managing this process in a reliable and efficient way

Version control: A component of software configuration management, version control, also known as revision control or source control, is the management of changes to documents, computer programs, large web sites, and other collections of information.



Especially important when collaborating with others

Why do we care?

Version control is an essential component in software development.

- Has been used by software developers for decades
- Source code lives in one or more repositories (repos) available to team members/contributors.
- Establishes a common context for code contributions and the exchange of ideas
- Establishes a chronological sequence of events
- Serves as "ground truth" for a software project
- Results from uncontrolled code are not *reproducible*.

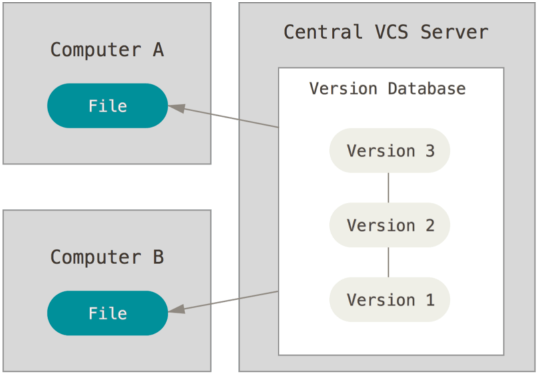
Why do we care?

Advantages of using version control

- A repository is your friend:
  - A repo can tell you exactly what version you are looking at (with a unique identifier).
  - Everyone can agree on whether they are looking at the same thing.
  - If there are conflicts, your version control system will tell you, and *you will need to resolve them*.

Centralized version control systems

There is one repository containing the master version (the ”trunk”) of the source code.

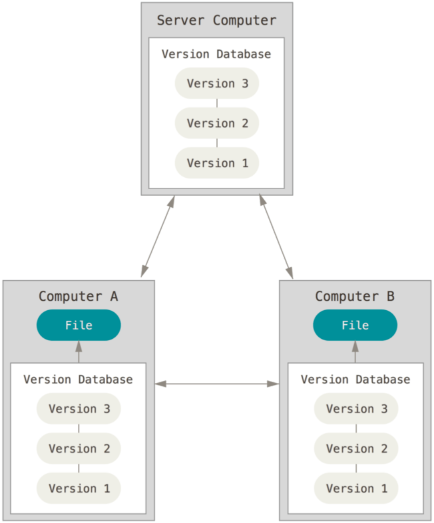


Distributed version control systems

Example: Git, Mercurial

- Everyone has a copy of the entire repo and its history(!)
- People typically work in development *branches*, with their changes isolated from others until merges are performed.
- Greater flexibility for design development procedures.
- Greater complexity (more concepts, fewer set rules)..

Distributed version control systems



https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Git Tutorial

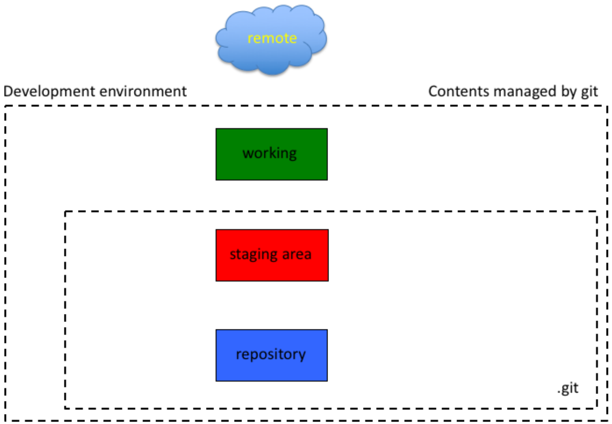
The Git concept

What is Git?[1]

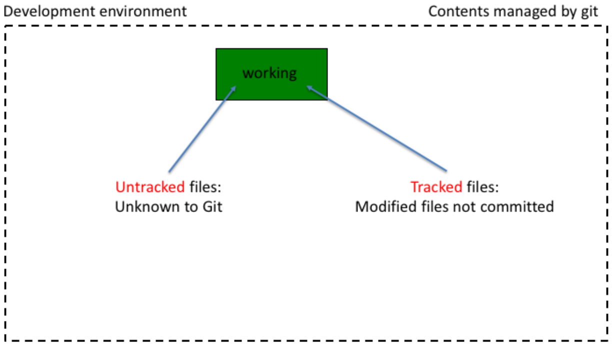
`Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.`

[1] https://git-scm.com

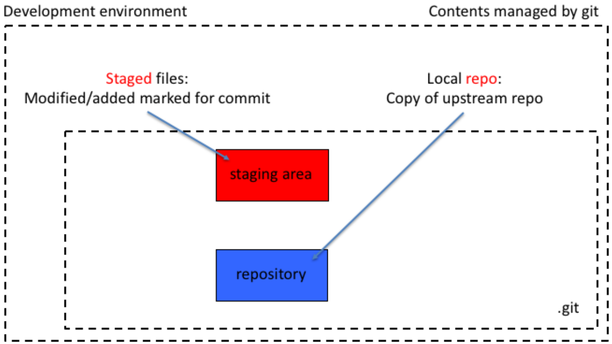
The Git Setup



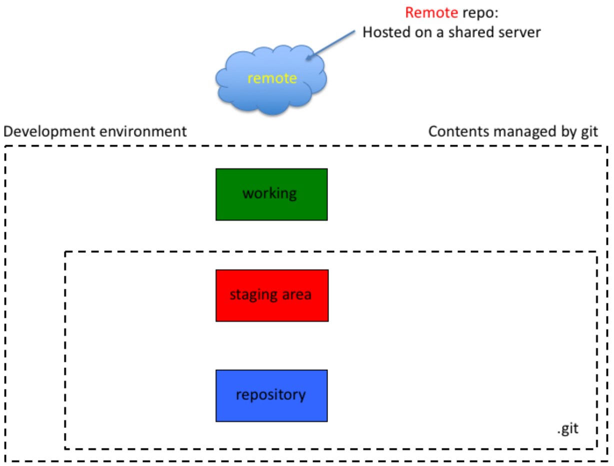
The Git Setup



The Git Setup

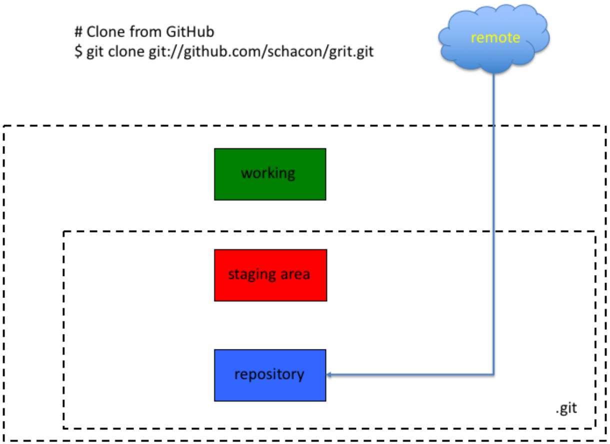


The Git Setup



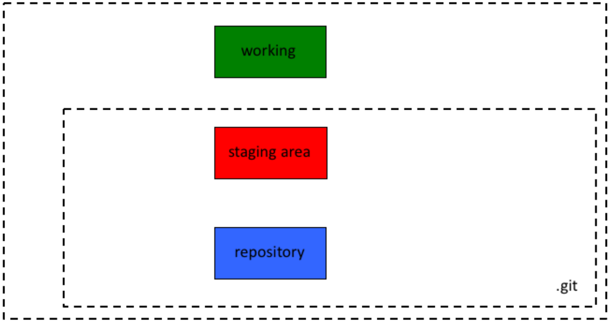
Setting up a Git repo

```
# Clone from GitHub
$ git clone git://github.com/schacon/grit.git
```



Setting up a Git repo

```
$ mkdir myproj
$ cd myproj
# Put myproj under version control
$ git init
Initialized empty Git repository in /some/path/myproj
```



Setting up a Git repo

Initializing a new project:

```
$ mkdir myproj
$ cd myproj
# Put my-proj under version control
$ git init
Initialized empty Git repository in /some/path/myproj
```

Initializing from an existing project:

```
# Clone from GitHub
$ git clone git://github.com/schacon/grit.git
```

Configuring Git

git-config - Get and set repository or global options

- System: /etc/gitconfig
- User: $HOME/.gitconfig
- Project: my_project/.git/config

Git commands to edit configuration

```
# System
git config --system
# User
git config --global
# Project
git config (project)
```

Configuring Git

Your identity:

```
git config --global user.name "Your name"
git config --global user.email you@domain.com
```

Your editor:

```
git config --global core.editor vim
```

Check settings:

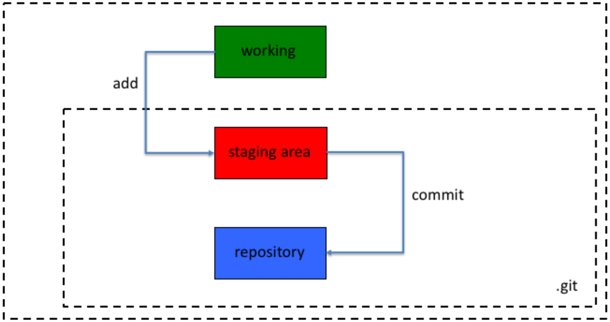
```
git config --list
```

Exercise 1

- Create a folder in the location of your choice, with the name of your choice, and initialize a git repository **in that folder**.
- Configure your git environment:
  - “git config user.name ”Your Name Comes Here” “
  - “git config user.email you@yourdomain.example.com“
  - “git config color.ui auto“
- Check your configuration settings by using “git config –list“

Making changes in the working directory

```
# Add a file
$ git add README
# Save a file
$ git commit –m 'Add a file' README
# Show commit log
$ git log
```



File Status

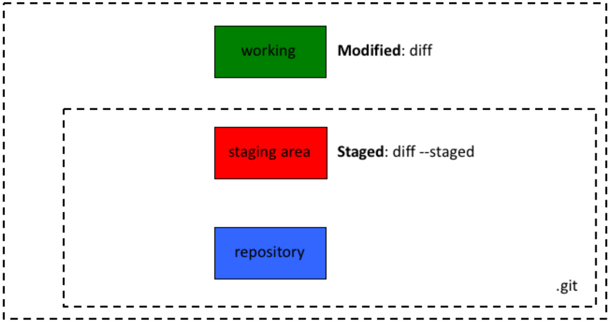
File in .git is a tracked file. A tracked file can be in 3 different states:

- modified
- staged
- committed

Untracked files: all other files

File Status

```
# Show commit log
$ git log
# Show status of repo
$ git status
# Differences
$ git diff
$ git diff --staged
```



Git workflow

So, in the morning:

```
$ git status
$ git log
# Edit a file, then
$ git diff [file] # shows changes between working dir and
    staging area
$ git add myfile
$ git commit -m'commit message' myfile
```

In the afternoon:

```
# After editing file again
$ git add myfile
$ git commit -m'different commit message' myfile
```

Exercise 2

- Create a new file “AUTHORS“ - with your name on it - and **add** it.
- Check what the status of your repository is, by using “git status“.
- Save the state of your repository with “git commit“
- Now edit the “AUTHORS“ file - e.g. add a co-author - , and check the status of the repository.
- Add this file and commit it.
- Type “git log“ in the repository. What do you see?
- type “git log –oneline –graph –decorate“ What do you see?

Deleting and moving files

```
# Remove files from working tree
$ git rm FILENAME
# Move a file or a directory
$ git mv FILENAME TARGET
# Put current changes into a stash
$ git stash
# Apply stash to working dir
$ git stash pop
# Clear stash
$ git stash drop
```

Canceling operations

```
# Unstage a file
$ git reset HEAD FILENAME
# Unmodify an unstaged file(s) - this operation in unrecoverable
$ git checkout filename [filename2,...]
# Get file back from staging area to working dir
$ git reset [file]
```

Exercise 3

- Rename the file “AUTHORS“ to “CONTRIBUTORS“ using “git mv“ and commit your changes
- Now delete this file and commit your changes
- Use “git log“ to see the history of the repository.
- Create a “TODO“ file, and add it to the staging area.
- Remove this file from the staging area.
- Create a python script called power.py
- Add this to the staging area and commit it.
- Now edit it again, and add the following function to the bottom of the file:

```
def square_root(x):
    return np.sqrt(x)
```

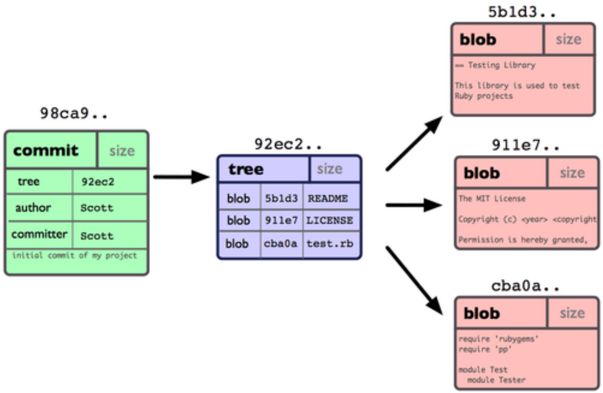
- Use git checkout to remove the changes you've made to this file. You can check what you have done using “git status“.

Git internals

For each change set, i.e. each commit:

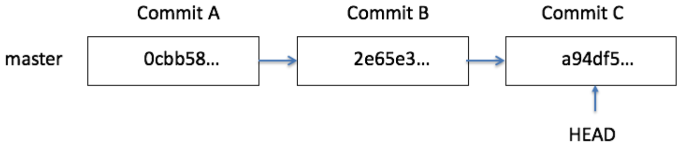
- Git generates a **checksum**
  - Git uses SHA1 algorithm to create checksums. SHA1 guarantees complete uniqueness.
  - 40-character hexadecimal string
  - e.g. 1fbb5af06b9e4facff4170fc687ecdd143daad50
- Git uses SHA1 string to organize database of references
- Fortunately, one can use a subset of SHA1 string (typically 4-8 characters)

Git internals



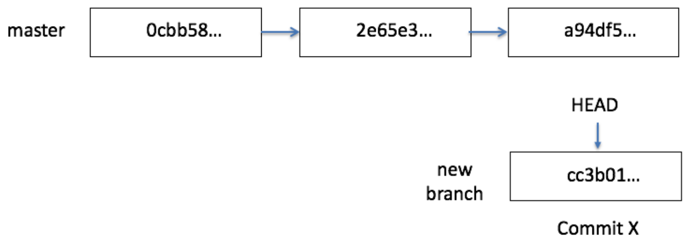
Branching

Commits are repository snapshots



Branching

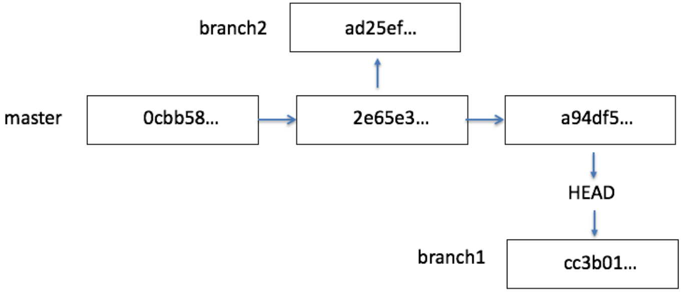
A branch is a pointer to commit - not a full copy.



HEAD points to tip of current branch in repository

Branching

Can have many branches



Demo

Branching

git-branch - Manage branches

```
# List current branch [or all]
$ git branch [-a]
# Create branch
$ git branch [name]
# Delete branch
$ git -d branch [name]
```

Branching

git-checkout - Move to a branch

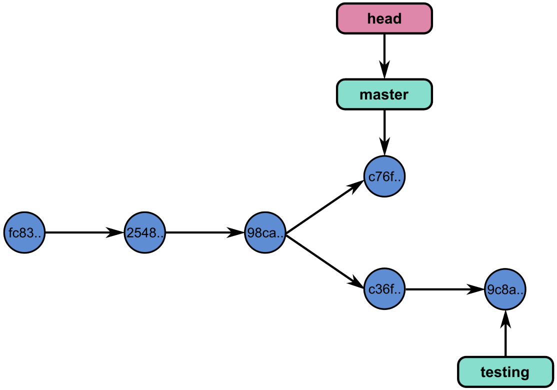
```
# Move to a branch
$ git checkout [name]
# Create and move to the branch in one step
$ git checkout -b [name]
```

Exercise 4

- Create a new branch “testing“.
- Switch to that branch.
- Check what branch you are in using “git branch“.
- Create a python script that prints the first 10 integers, and commit it.
- Look at the history of your repository.
- Switch to the branch “master“, and look again at the history. What do you see?

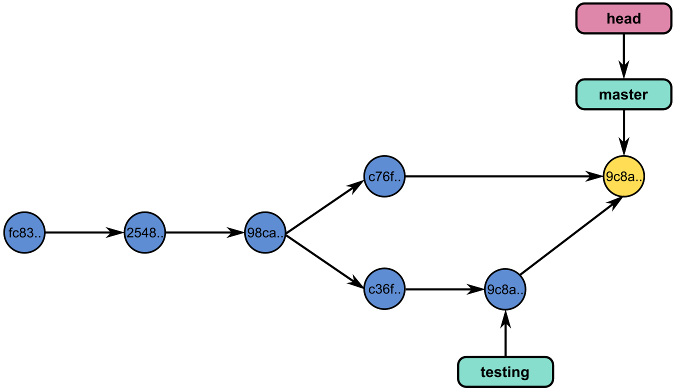
Merging

We have two branches



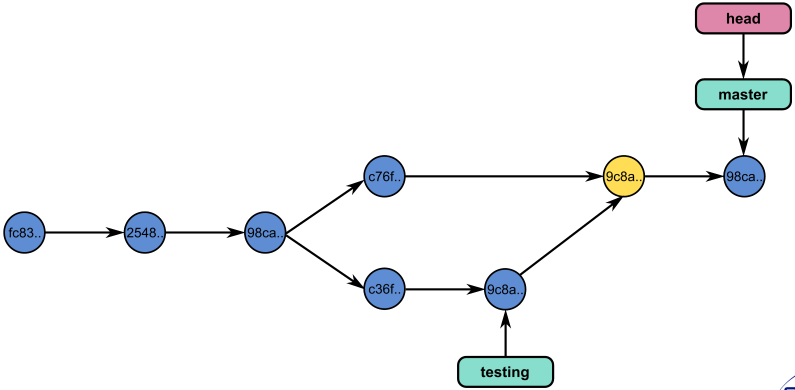
Merging

Merge test unto master



Merging

Continue working (on either branch) or delete branch if done



Merging

```
# Merge two branches
$ git merge
# Merge specific branch
$ git merge [name]
```

When working with a remote repo you "need" to **fetch** branches. "fetch" downloads new data from a remote repository - but it doesn't integrate any of this new data into your working files.

```
# Git fetch
$ git fetch origin
# Inspect branch
$ git merge
```

pull = fetch + merge;

```
# update your current HEAD branch with the latest changes from
    the remote server
$ git pull
```

Merge conflicts

When a conflict emerges, you must manually edit the files

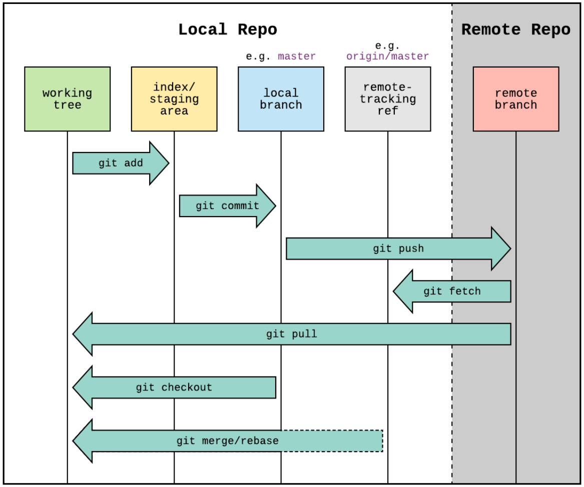
```
<<<<<<< HEAD:calc.py
print 'the average is', sum(x) / float(len(x))
print 'sumsqdiffs is', sum(diffs)
=======
avg = sum(x) / float(len(x))
sumsqdiffs = sum(diffs) / float(len(x))

print 'average is', avg, 'and sumsqdiffs is', sumsqdiffs
>>>>>>> edgier:calc.py
```

Exercise 5

- Try deleting branch “testing“ using “git branch -d testing“. What do you see?
- Merge the changes of branch “testing“ in “master“.
- Try to delete the branch “testing“ once again.

Git workflow



Working with a server

Remotes

Remote repositories are versions of your project that are hosted on the Internet or network somewhere.

```
# list the remote servers you have configured. Tip: For more
    verbosity, add -v option.
git remote list
# adds the [url] as a remote
git remote add name url:
# remove the remote [name]
git remote rm [name]:
```

Collaborating with GitHub

GitHub is a web-based hosting service for version control using Git. It is mostly used for computer code. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features.[2]

http://www.github.com

[2] https://www.wikiwand.com/en/GitHub

GitHub

- Free repositories for Open Source projects
- Implements several helpful process "building blocks"
- Includes some simple niceties (issue tracker, wiki, pretty log/graph visualizations)
- Other similar services exist (Bitbucket, GitLab, ?)

Github

Updating a repository

```
# fetches the branches on the remote. The branches from that
# remote are then accessible locally in
# [remote-name/branch-name]
git fetch [remote-name]
# pushed [branch-name] onto remote [branch-name]
git push [remote-name] [branch-name]
# merges [branch-name] into the current branch
git merge [branch-name]
```

Exercise 6

- Create a github project.
- Add the url of the project as a remote called “my_repository“.
- Push your changes to github: “git push my_repository master“
- Check on github that you have indeed pushed your changes.
- Fetch the changes from the remote “origin“.
- Merge the changes from “origin/master“ to your local master branch (there may be a conflict - if so, resolve it).
- Push the new changes to your remote called “my_repository“.

Resources

- Git Community Book
- Pro Git
- A Visual Git Reference

The End

Questions?