



git



SciComp git for Bioinformaticians V2.2.1

1 git Logo Courtesy Jason Long; licensed under Creative Commons Attribution 3.0 Unported License
gitlab fox logo Courtesy GitLab B. V.

Agenda

- What is version control?
- My first repository
- Basic commands
- Intermediate commands
- Exercises for the reader

What is Version Control and
why do I need it?

3

Git started

4

(non exhaustive) Approved Git Clients

- Stand Alone Clients
 - Git
 - Git for Windows (msysgit)
 - Tortoise Git
 - SourceTree
- IDEs with Git integration
 - Eclipse (Java, C/C++, others)
 - NetBeans (Java, C/C++, others)
 - R Studio (R language)

5

Make your client Git you

Right now!

Open a terminal.

```
$ cd
$ git config --global user.name <your username>
$ git config --global user.email <your email>
$ git config --list
$ git config user.name
$ git config user.email
```

6

Git setup

- Gitlab is a webserver that helps you create and manage git repositories.
- The client runs on your computer managing local repositories and connects to gitlab.
- Two kinds of connections
 - ssh
 - http(s)

7

Git on it!

Log into our training Gitlab instance by pointing a web-browser to:

<http://training.biotech.cdc.gov/>

Your username is on the little piece of paper with your training packet.

8

Your SSH Public Key

- The key is used to prove you are who you say you are (authentication)
- For this to remain valid no one else should know your private key (the one that does not end in .pub)
- Your client needs to know your private key
- The server needs to know your public key

9

Git a pair

ssh-keygen

- Use the defaults
- Don't set a passphrase

10

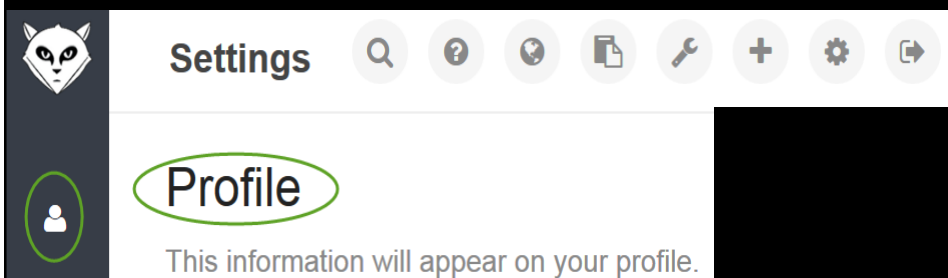
Telling gitlab who you are.

Adding public key to your gitlab profile:

Follow the leader!

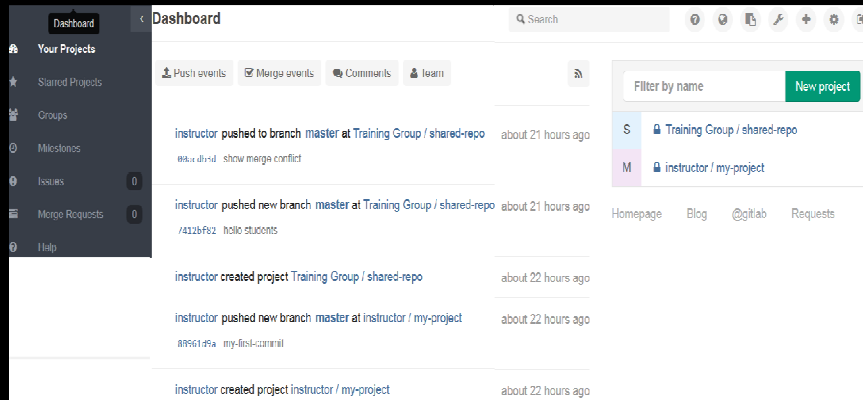
11

Click Profile Settings



12

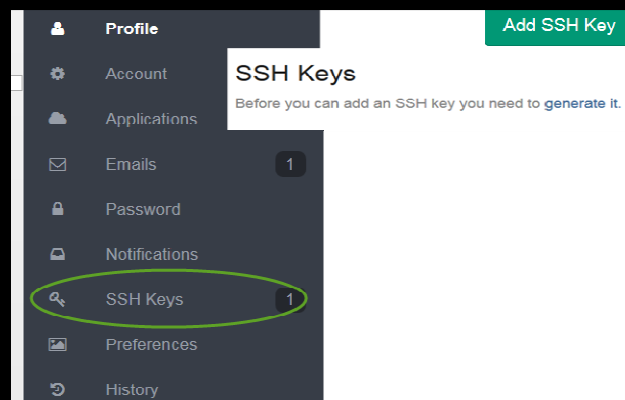
Dashboard



13

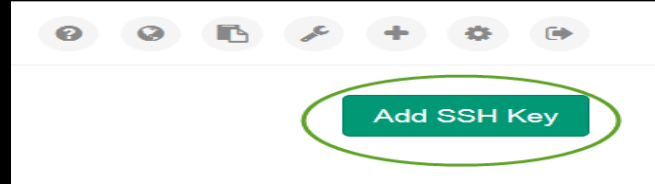
Click SSH Keys

An SSH key allows you to establish a secure connection between your computer and GitLab .To generate a new SSH key, just open your terminal and type ssh-keygen. Press enter to generate one.



14

Click Add SSH Key



Title	Fingerprint
<code>instructor@scicomp-train-01.biotech.cdc.gov</code>	<code>0b:eb:07:a9:bf:0c:a1:d4:e5:dc:be:a5:3a:31:f0:71</code>

Added at
added about 2 hours ago Remove

15

Copy your ssh pub key

- You need to copy the contents of `~/.ssh/id_dsa.pub` into the key field.
- Make sure there are no newlines it should all be one long line.
- The title is just so you can remember what the key is for.

16

Some important definitions to Git.

■ Repository (AKA: Repo)

1. A conceptual place where all of the history of a set of tracked files is kept.

■ Commit

1. A set of changes, to tracked files, that are stored as a unit in a repository.
2. to store a set of changes in a repository.

■ Change

1. Creation, deletion, editing, renaming, and alteration of permissions on a file.

17

Git'n into git nirvana.

■ Working Directory

The directory where you make changes to files. File changes in the working directory can be added to the Staging Area.

■ Staging Area (AKA: Index, Staged Files)

A place where you assemble and groom a commit before storing it in your local repository.

■ Local Repository

The local repository is a repository on the computer you are working on.

■ Remote Repository

A remote repository is a repository hosted on a server. In general this refers to a Repository that tracks the same set of files, and is a copy of the local repository.

18

The staging area

Usual version control systems provide two spaces:

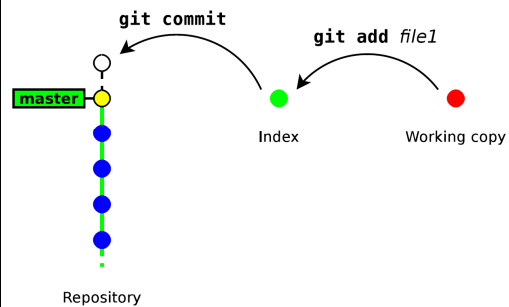
- the **repository**
(the whole history of your project)
- the **working tree** (or **local copy**)
(the files you are editing and that will be in the next commit)

Git introduces an intermediate space : the **staging area**
(also called **index**)

The index stores the files scheduled for the next commit:

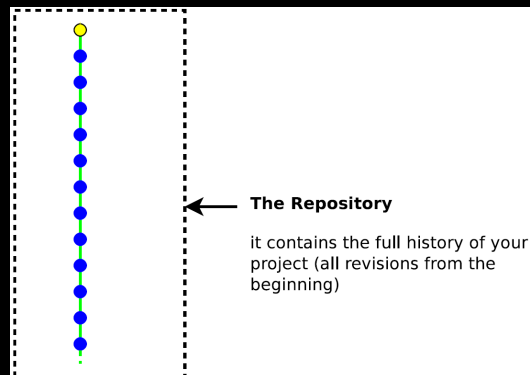
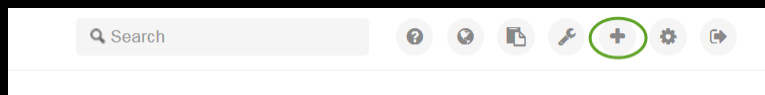
- `git add files` → copy files into the index
- `git commit` → commits the content of the index

The staging area



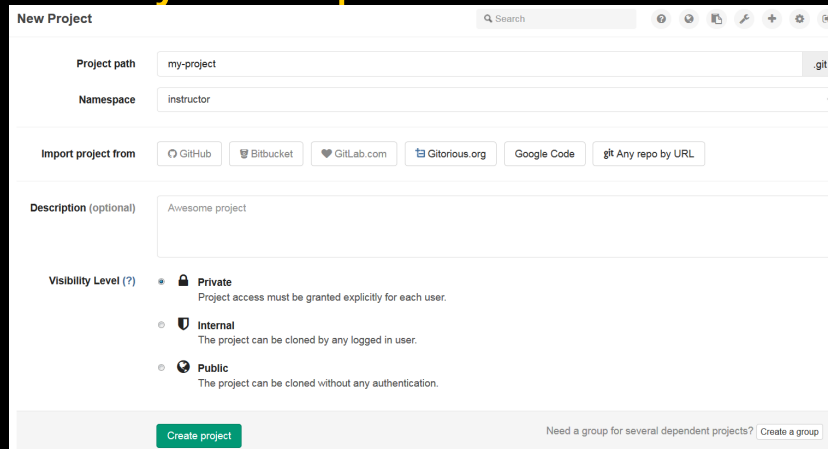
19

Git a repository or project



20

Name your repo



The screenshot shows a 'New Project' form with the following fields and options:

- Project path:** my-project
- Namespace:** instructor
- Import project from:** GitHub, Bitbucket, GitLab.com, Gitorious.org, Google Code, git Any repo by URL
- Description (optional):** Awesome project
- Visibility Level (?):**
 - ☒ **Private**
Project access must be granted explicitly for each user.
 - ☐ **Internal**
The project can be cloned by any logged in user.
 - ☐ **Public**
The project can be cloned without any authentication.

At the bottom, there is a green 'Create project' button and a link 'Need a group for several dependent projects? Create a group'.

- Give your new baby the name: 'my-repo'
- Click 'Create Project'

21

Clone your new repo

git clone makes an exact copy of another repository including all its history.

SSH	HTTP	git@scicomp-train-01.biotech.cdc.gov:Instructor/my-repo.git
-----	------	---

Right now!

```
$ git clone <your url>
```

22

Basic git commands

23

status

- `git status` → show the status of the index and working copy

What's going on here?

Right now!

```
$ cd ~/my-repo
$ git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use
"git add" to track)
```

24

add

- `git add files` → copy files into the index

Stage your changes

Right now!

```
$ cat > hello.py
print 'hello world!'
<ctrl+d>
$ python hello.py
hello world!
$ git add hello.py
$ git status
```

- “`git add . && git commit`” commits everything in the working tree (including new files)

What just happened?

25

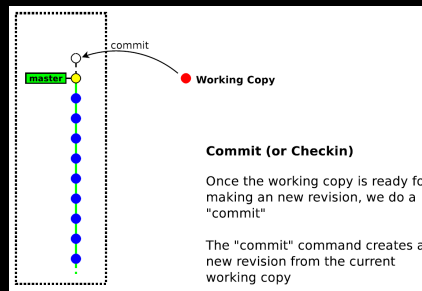
commit

Submits your staged changes to your local repository

Right now!

```
$ git commit -m 'My first commit!'
[master (root-commit) 055731c] My first commit!
```

`git commit`
→ commit the index



26

push

Pushing (uploading) local changes to the remote repository

Make changes in your repository
available on a remote.

Right now!

```
$ git push origin master
```

** Now go to your repository on the gitlab site

Now go to the page on gitlab for your my-repo
repository.

27

Now we're Git'n jiggy with it!

- clone, add, commit, and push represent the basic git workflow.
- They always happen in that order.

28

init

Create empty local repository

Right now!

```
$ cd
```

```
$ git init shared-repo
```

```
Initialized empty Git repository in  
/scicomp/home/instructor/shared-repo/.git/
```

```
$ cd shared-repo
```

```
git init myrepository
```

This command creates the directory *myrepository*.

- the repository is located in *myrepository/.git*
- the (initially empty) working copy is located in *myrepository/*

29

Git with friends

30

remote

```
git remote add name
```

name is a local alias identifying the remote repository

Manage connections with remote repositories

Right now!

```
$ git remote add origin  
gitlab@training.biotech.cdc.gov:training/shared-repo.git  
$ git remote  
$ git remote --verbose
```

How git handles remote repositories

- Remote repositories are mirrored within the local repository
- It is possible to work with multiple remote repositories
- Each remote repository is identified with a local alias. When working with a unique remote repository, it is usually named **origin**¹¹
- Remote branches are mapped in a separate namespace: `remote/name/branch`. Examples:
 - `master` refers to the local master branch
 - `remote/origin/master` refers to the master branch of the remote repository named origin

31

pull

```
git pull merges immediately the remote branch into the current local branch.
```

Grab changes from a remote and combine them with your working directory.

Right now!

```
$ git pull origin master  
$ python hello.py  
$ git status
```

32

Gitting a bad pull

Sometimes when two people are editing the same file git will not know how to combine both sets of changes.

Right now!

Edit hello.py to say hello instructor!

```
$ python hello.py
```

```
hello instructor!
```

```
$ git add hello.py; git commit -m 'message'
```

```
$ git pull origin master
```

Auto-merging hello.py

CONFLICT (content): Merge conflict in hello.py Automatic merge failed; fix conflicts and then commit the result.

33

Git back on that horse

Edit hello.py

```
<<<<<< HEAD
print 'hello instructor!'
=====
print 'hola students!'
>>>>>> b2c472ef20141ef5b2b6194ae93e72e2ae63a06f
```

■ Git will refuse to commit the new revision until all the conflicts are explicitly resolved by the user

Right now!

```
$ git status
```

Edit hello.py to only have your content.

```
$ git add hello.py; git commit -m 'resolved'
```

34

Resolving conflicts

Resolving conflicts

There are two ways to resolve conflicts:

- either edit the files manually, then run

```
git add file → to check the file into the index
or
git rm file → to delete the file
```

- or with a conflict resolution tool(xxdiff, kdiff3, emerge, ...)

```
git mergetool [ file ]
```

Then, once all conflicting files are checked in the index, you just need to run `git commit` to commit the merge.

35

checkout

```
git checkout -- path
```

This command restores a file (or directory) as it appears in the index (thus it drops all unstaged changes)

Replace anything in your Working Directory with what it looks like in your Repository.

Right now!

Edit `hello.py` to say:

```
lprint 'hello world!'
```

```
$ python hello.py
```

```
$ git checkout hello.py
```

```
$ python hello.py
```

36

reset

Un-stage changes (opposite of add)

Right now!

```
$ nano hello.py
**change the print to say hello to yourself.
$ python hello.py
hello instructor!
$ git add hello.py
$ git status
$ git reset hello.py
$ git status
```

`git reset` cancels the changes in the index (and possibly in the working copy)

- `git reset` drops the changes staged into the index⁷, but the working copy is left intact

37

rm

```
git rm file
```

→ remove the file from the index and from the working copy

- Mark file as deleted from repository
- Effects take effect going forward the file still exists in the repository history.

Right now!

```
$ touch delete-me.py
$ git add delete-me.py; git commit -m 'abc123'
$ git status
$ git rm delete-me.py
$ git status
$ git commit -m 'removed file'
$ git status
```

38

Intermediate git commands

39

First, what is a branch and tag?

- Branch is a parallel history for a repository
- Tag is a sign post for a particular point in history
- The HEAD is the git term for the current position (branch and place in repo history) you are on.

40

tag

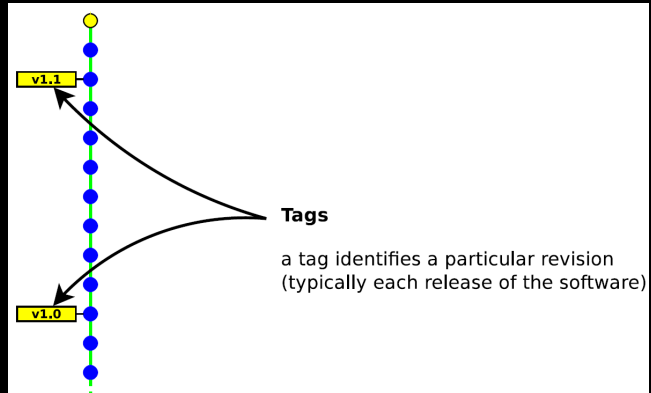
- `git tag` → creating/deleting tags (to identify a particular revision)

Plant a flag.

Right now!

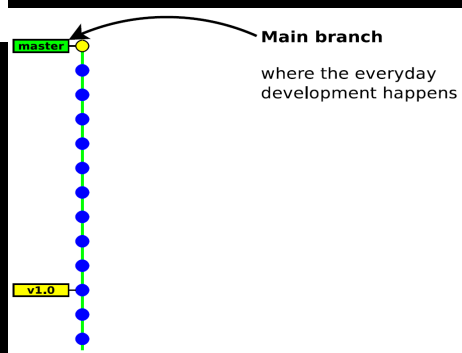
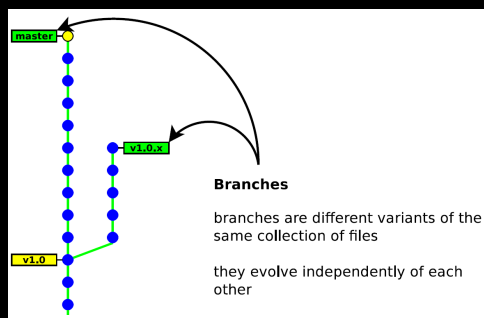
```
$ git tag my-first-tag
```

```
$ git tag
```



41

Branch



42

branch → a **branch** is just a pointer on the latest commit.

Create a parallel part of the repo tree

Right now!

```
$ git checkout -b my-first-branch
Switched to a new branch 'my-first-branch'
$ git status
$ git branch
  master
* my-first-branch
$ git checkout master
Switched to branch 'master'
```

43

log • `git log` → show the history

Show me the commit messages.

Right now!

```
$ git log | less
```

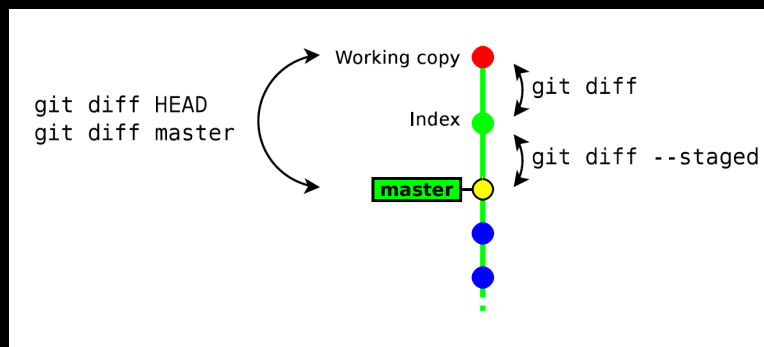
44

Git 'R Done! Exercises for the reader

45

diff → shows the differences

- How are two parts of history different (including now).



46

stash

- Take changes you have made but not committed and save them for later.
- Makes all files the same as they are in the HEAD but does not discard the changes.

47

fetch and merge

In practice, it is more convenient to use `git pull`, which is an alias to `git fetch + git merge`

```
git pull
```

- Strictly speaking a pull is a fetch followed by a merge.
- Fetch gets changes from the repo but does not update your working area.
- Merge combines arbitrary histories in your working directory.

48

rebase

- Become the ministry of information and rewrite history.

49

Gitlab permissions and repo types

- public, private, internal.
- Adding users to your own projects

50

Workflows

- Too complex to cover here (you need some experience with git before it would be helpful).
- Sensitive to local conditions and personal preferences
- <http://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows>

51

Questions?

52

Myriad of resources for git

- git <command> --help
- Official git documentation: git-scm.com
- Github documentation: github.com
- google it!
- Tutorials
 - official tutorial <http://git-scm.com/docs/gittutorial>
 - <https://www.atlassian.com/git/tutorials/>
 - SciComp Screen casts for setting up various clients:
http://info.biotech.cdc.gov/?page_id=56

53

Gitting Help!

- Ask the SciComp Support Team!
 - Email: OAMDSupport@cdc.gov
 - Great for questions
 - Great if you don't know if it's a real error
 - Email: scicompssupport@mail.biotech.cdc.gov
 - Auto submits a trouble ticket to our helpdesk
 - Great if you *know* that you have a an issue
 - NOT great for questions. Use other email.

Now Git to it!

54