UT Southwestern Medical Center
Lyda Hill Department of Bioinformatics

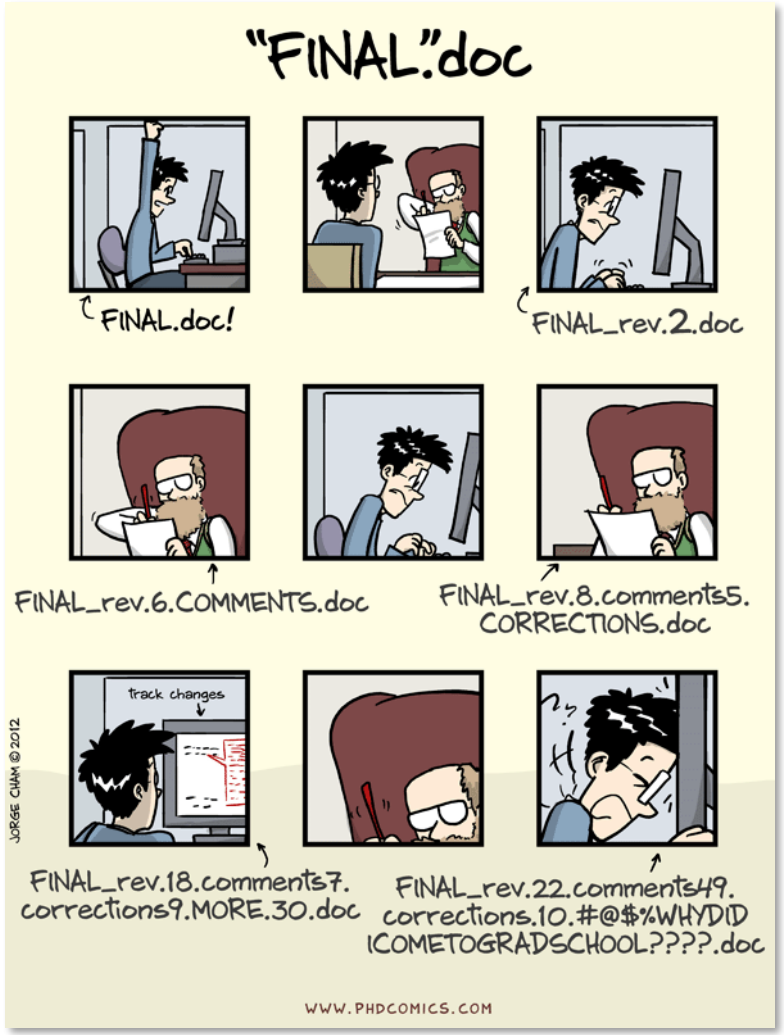
BioHPC

Introduction to Git Version Control System

Contributions from:
Venkat Malladi
Wei Guo

August 2021

Why version control systems exist...



Bad enough for a single manuscript!

Imagine the same for a code base...

Don't confuse a VCS with a backup or a deployment system. You don't have to change or replace any other part of your tool chain when you start using version control.

A VCS simply records the changes you make to your project's files.

"Piled Higher and Deeper" by Jorge Cham
www.phdcomics.com

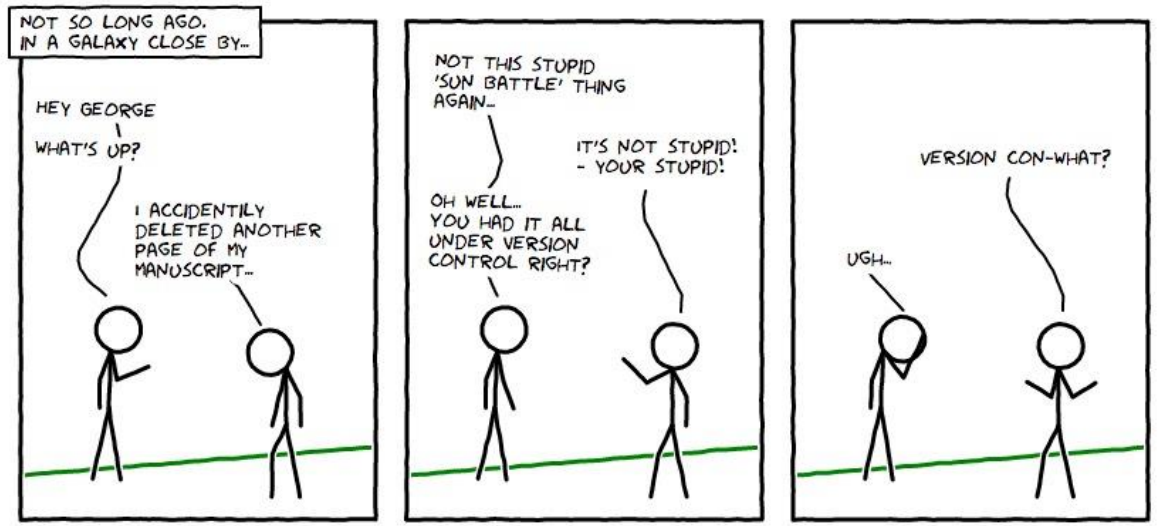
Outline

I. Introduction to git
   A. What is git?
   B. Git workflow: creating a new repository
   C. Head
   D. Basic git commands
   E. Concept of branches
   F. Creating a branch/switching between branches
   G. Merging branches and resolving conflicts

II. Introduction to Gitlab
   A. What is GitLab?
   B. GitLab in practice: distributed version control
   C. Cloning a remote repository
   D. Fetching/pushing to a remote repository
   E. Collaborating using git and GitLab

What is a “version control system” and what is a repository?

- A way to manage files and directories.
- Track changes over time.
- Recall previous versions.
- Repo is short for repository.
- Usually used to organize a single project.
- Repos can contain folders and files, images, videos, spreadsheets, and datasets – anything your project needs.

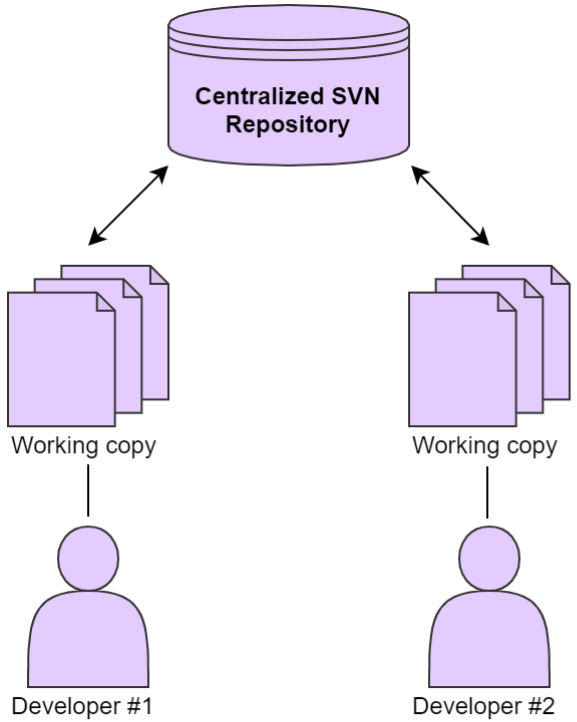


In git, the repository is just a simple hidden folder named ".git" in the root directory of your project.

What is git?

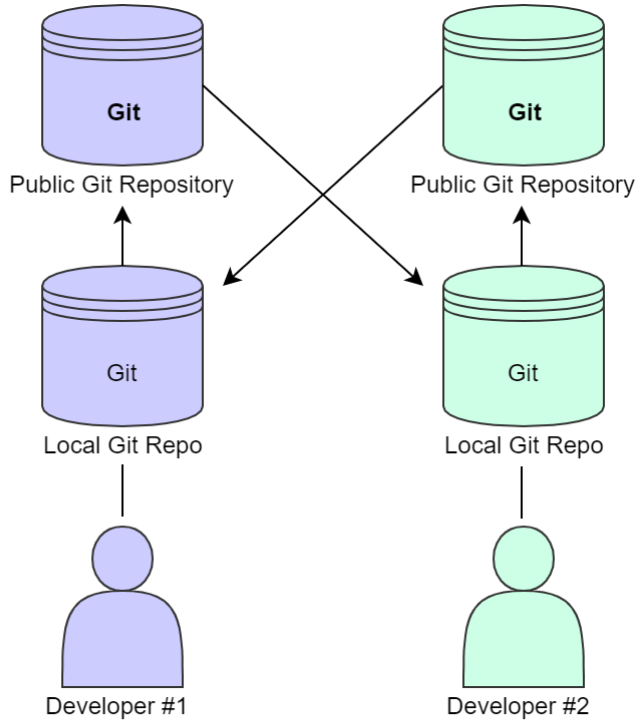
- Created by Linus Torvalds in 2005.
- A command line version control program.
- Uses checksums to ensure data integrity.
- Cross-platform.
- Open source, free.
- Distributed version control (as opposed to centralized).

Distributed version control



Centralized Development

There is one central repository and the developers (no matter how many of them there are) can commit only to that repo.



Distributed Development

The repositories are interconnected and can be edited and worked on by several developers simultaneously.

https://i.ibb.co/cgCdpyH/Central-Distributed.png

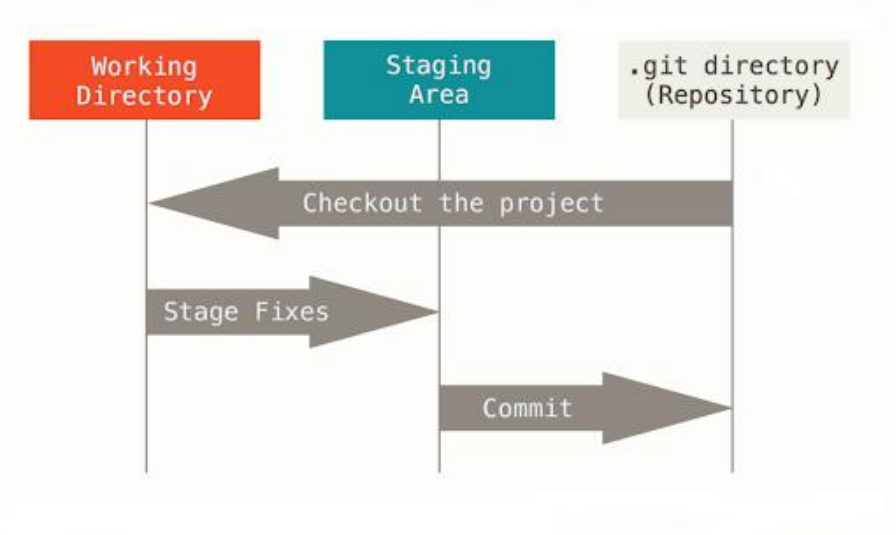
Advantages of distributed version control

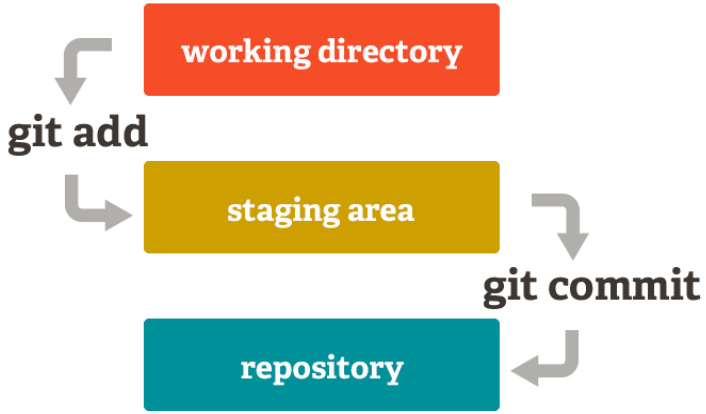
- No need to connect to central server.
- Can work without internet connection.
- No single point of failure.
- Developers can work independently and merge their work later.
- Every copy of a git repo has the **complete** history.

Git tree architecture

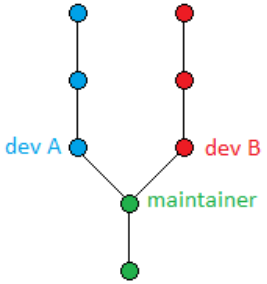
Basic git workflow:

- Modify files at the working directory.
- Stage files by adding snapshots to staging area.
- Commit the changes to the git directory.

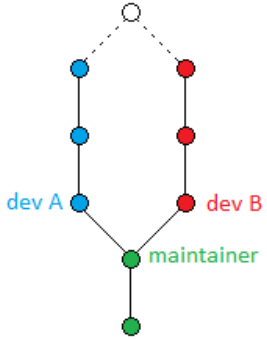




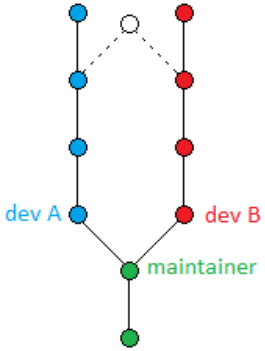
Git tree and use cases



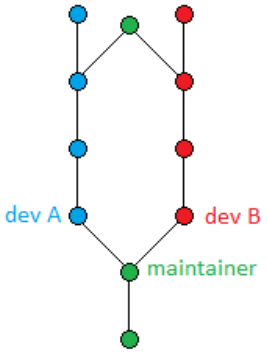
Both devs make commits



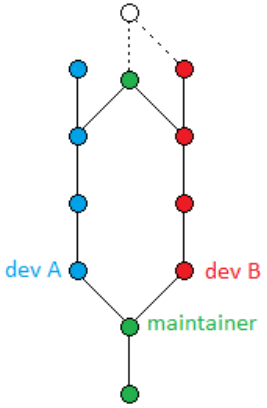
Both devs ask maintainer to push their commits



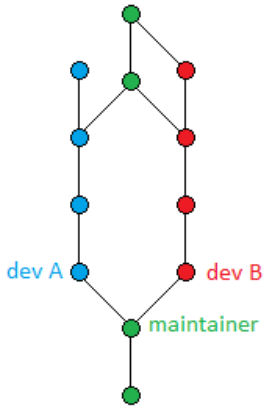
Both devs continue their work



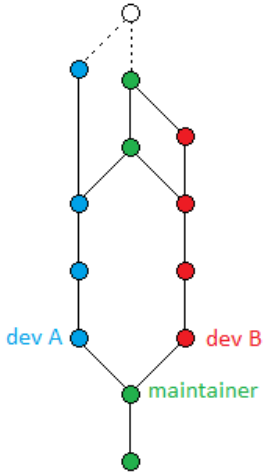
Maintainer adds their commits and merge it



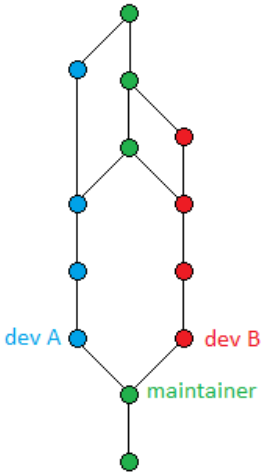
Dev B asks maintainer to push his commit



Maintainer merges his commits



Dev A asks maintainer to push his commit



Maintainer merges his commits

Users with the Developer role can create a project in a group but might not be allowed to initially push to the default branch.

https://i.stack.imgur.com/guGTo.png

Before starting to use git

- Setup your name and email so others can know who committed changes:

```
$ git config --global user.name s191529
$ git config --global user.email daniela.daniel@utsouthwestern.edu
```

Note: set for all repositories on your computer.

```
$ git config --local user.email daniela.daniel@utsouthwestern.edu
```

Note: set differently for each repository.

```
$ git config --list
user.name=s191529
user.mail=daniela.daniel@utsouthwestern.edu
user.email=daniela.daniel@utsouthwestern.edu
push.default=simple
```

A simple git workflow

1. Initialize a new project in a directory (this creates a new subdirectory named .git that contains all of your necessary repository files):

```
$ git init
```

2. Add a file using a text editor to the directory.

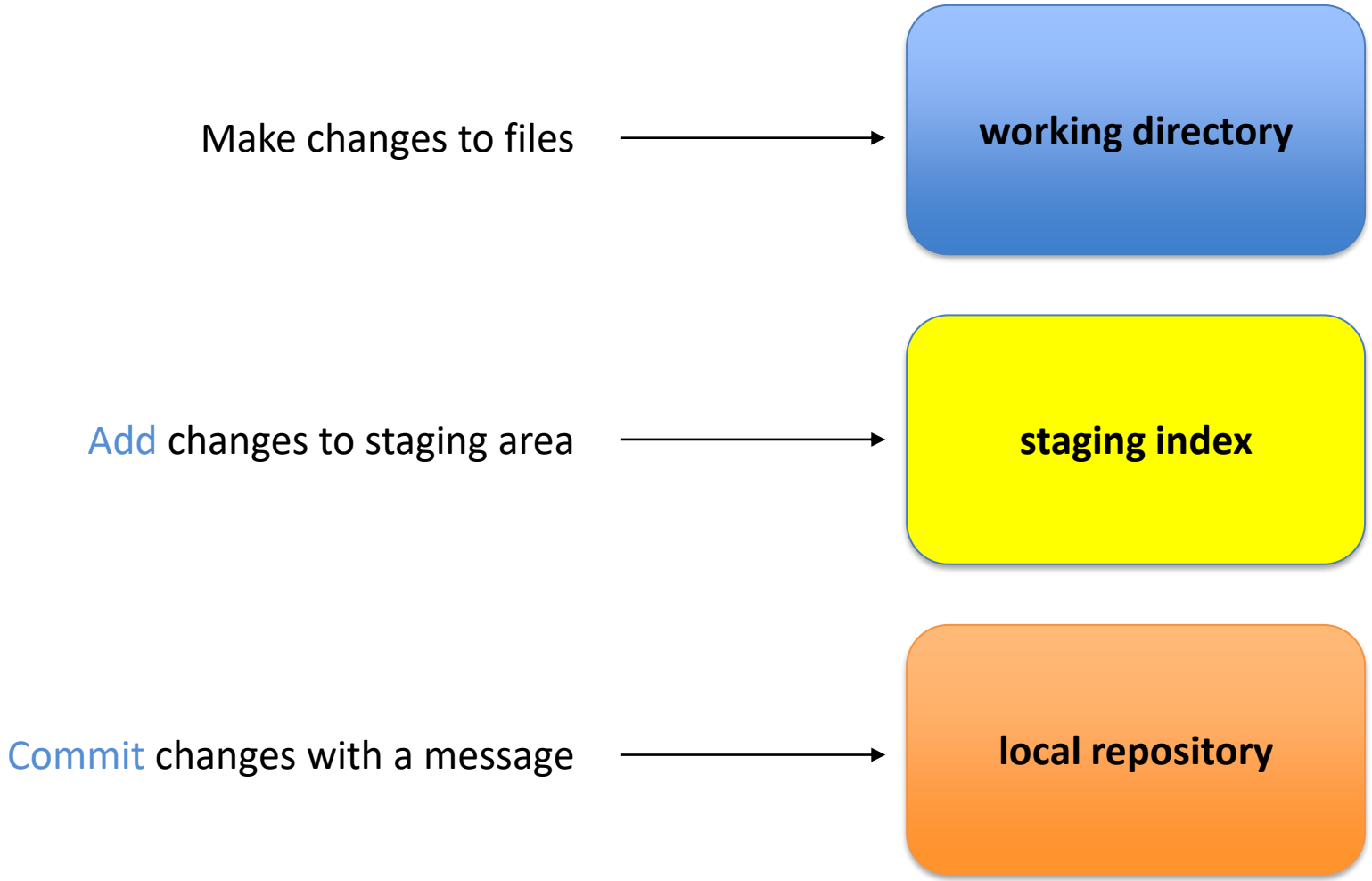
3. Add every change that has been made to the directory:

```
$ git add hello.txt
```

4. Commit the change to the repo:

```
$ git commit -m "important message here"
```

After initializing a new git repo



Demo: Initializing a new repository

```
$ mkdir learning_git
$ cd learning_git
$ git init
Initialized empty Git repository in
/endosome/work/biohpcadmin/s191529/git/learning_git/.git/
$ touch foo.txt
$ git add foo.txt
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   foo.txt
#
$ git commit -m "initial commit"
[master (root-commit) 44e7c64] initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 foo.txt
```

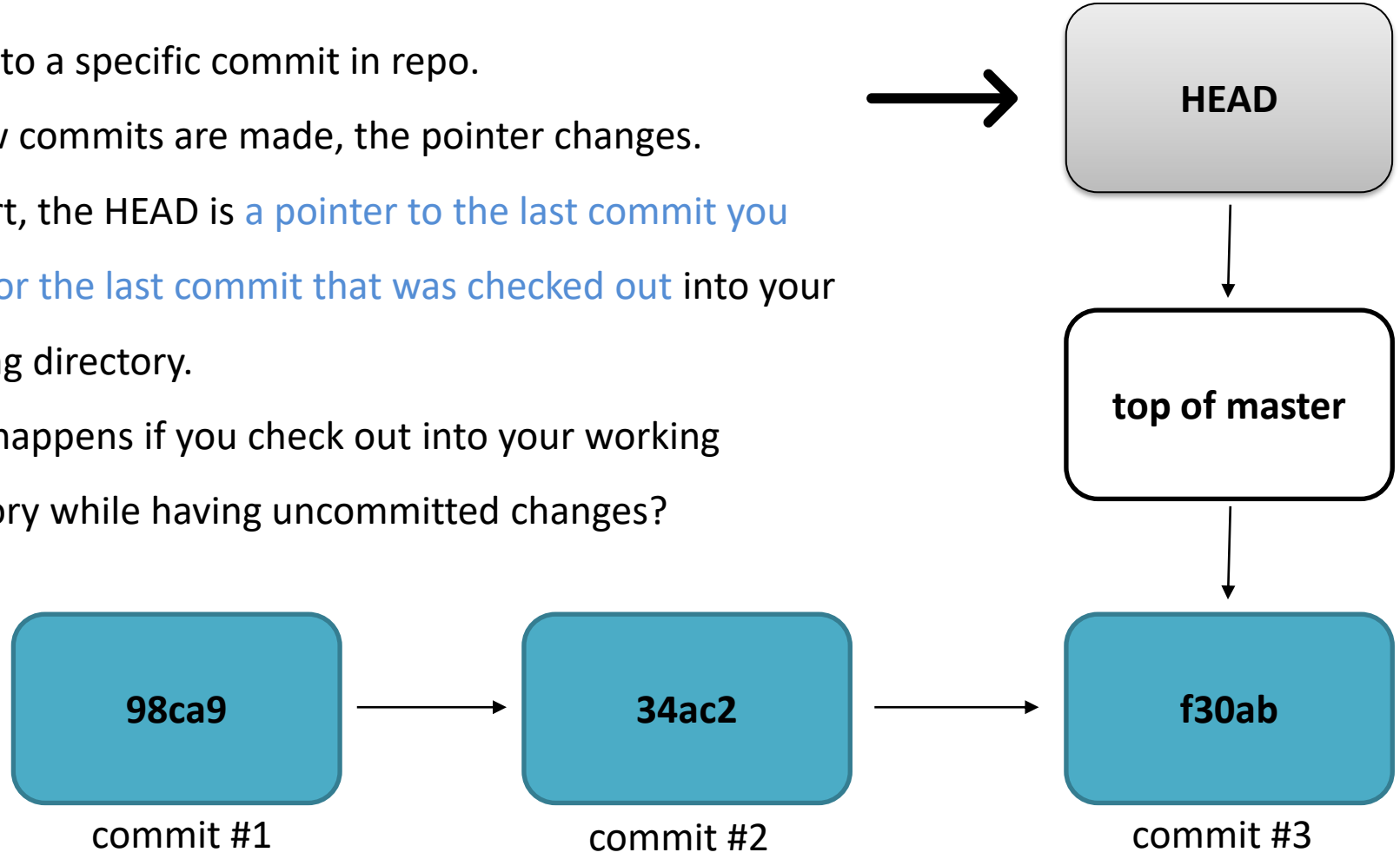
Commit messages

- Tell what it does (present tense).
- Single line summary followed by blank space followed by more complete description.
- Keep messages shorter than 72 characters.
- Ticket or bug number helps.

```
$ git log
commit 44e7c640286f4ac758670f7a39e145533a14b8c3
Author: s191529 <daniela.daniel@utsouthwestern.edu>
Date:   Fri Dec 4 15:03:33 2020 -0600

    initial commit
```

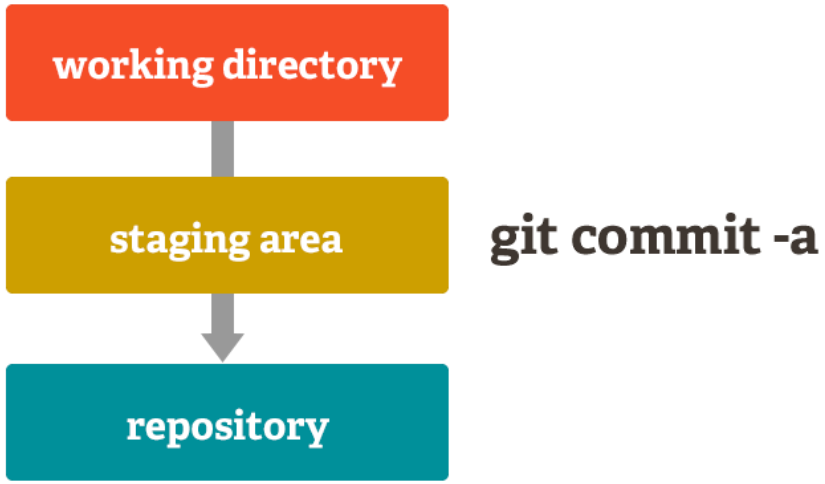
The HEAD pointer

- Points to a specific commit in repo.
- As new commits are made, the pointer changes.
- In short, the HEAD is a pointer to the last commit you made or the last commit that was checked out into your working directory.
- What happens if you check out into your working directory while having uncommitted changes?



Committing all changes of tracked files

- Allows one to add to staging index and commit at the same time.
- Grabs everything in working directory.
- Files not tracked or being deleted are not included:
  - *Files are tracked automatically after they are staged for the first time*



What changes were made?

`$ git diff` compares changes to files between repo and working directory

```
$ echo "adcdefghijklmnopqrstuvwxyz" >> foo.txt
$ git diff
diff --git a/foo.txt b/foo.txt
index e69de29..ab74472 100644
--- a/foo.txt
+++ b/foo.txt
@@ -0,0 +1 @@
+adcdefghijklmnopqrstuvwxyz
```

Note: `git diff --staged` compares staging index to repo

Note: `git diff <filename>` can be used as well

Difference between commits

```
$ git diff <commit> <commit>
```

When using checksum of older commit, will show you all changes compared to those in your working directory

```
$ git diff 44e7c640 a0c334ed
diff --git a/foo.txt b/foo.txt
index e69de29..ab74472 100644
--- a/foo.txt
+++ b/foo.txt
@@ -0,0 +1 @@
+adcdefghijklmnopqrstuvwxyz
```

Deleting files from the repo

`$ git rm <file>` moves deleted file change to staging area (changes still need to be committed)

```
$ touch bar.txt
$ git add bar.txt
$ git commit -m "bar.txt"
$ git rm bar.txt
rm 'bar.txt'
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       deleted:    bar.txt
#
$ git commit bar.txt -m "removed bar.txt"
```

Moving or renaming files

```
$ git mv <file> <file>
```

```
$ git mv foo.txt alphabet.txt
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       renamed:    foo.txt -> alphabet.txt
#
```

Cloning a repository

`$ git clone <repo>` or `git clone <repo> <repo>`

Clones a repo and all its branches

Does **not** clone uncommitted changes in the working directory

A `git clone --recursive <repo>` will also clone all git sub-repositories in a repository

```
$ cd ..
$ git clone learning_git test_git
Cloning into 'test_git'...
done.
$ cd test_git/
$ cat alphabet.txt
cat: alphabet.txt: No such file or directory
```

Clone to another directory

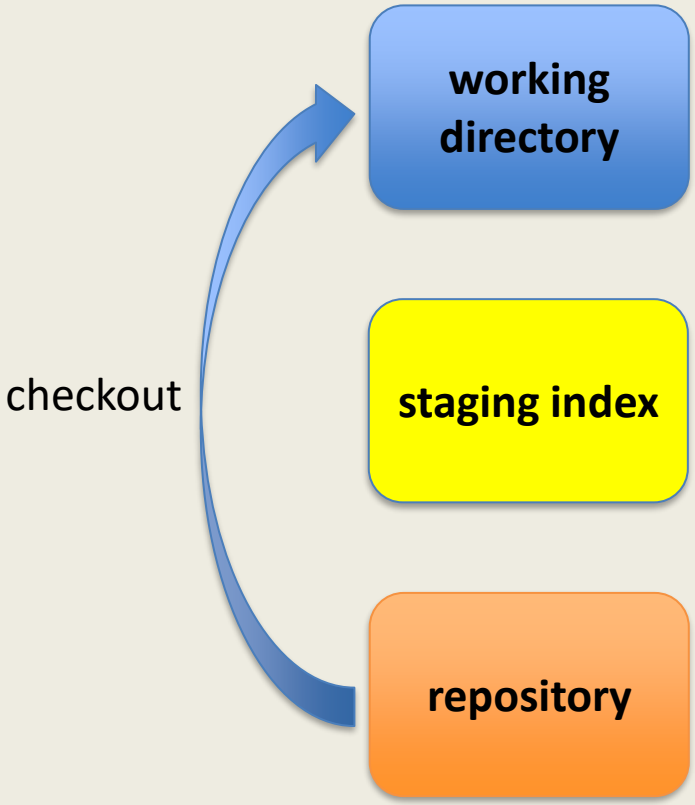
Why there is no `alphabet.txt` in this clone?

Undo changes made to a working directory

`git checkout <file>` will grab the file from the repo removing all changes since last commit

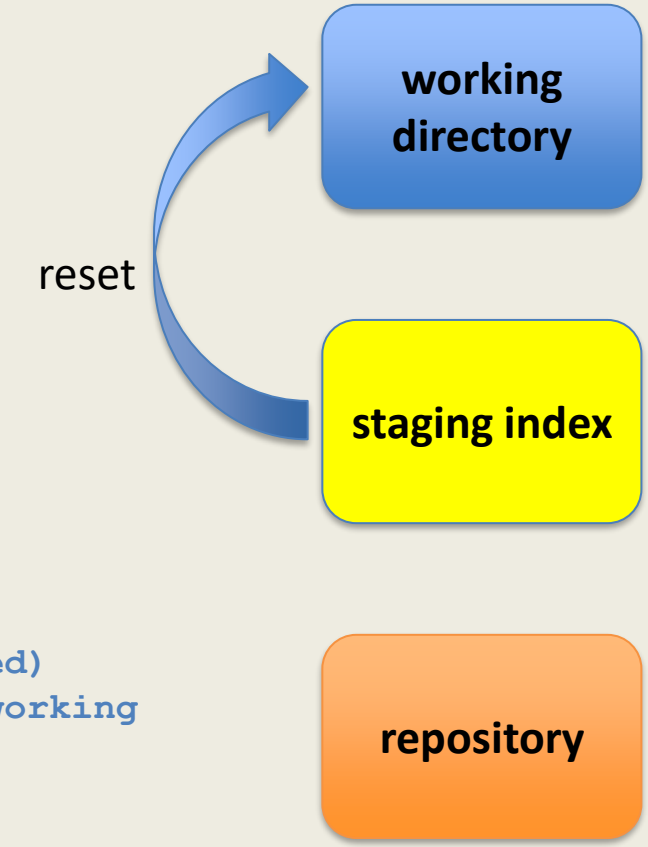
Let's commit the changes in the original repo:

```
$ cd ../learning_git/
$ git commit -a -m "renamed foo.txt"
[master ab9bffd] renamed foo.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename foo.txt => alphabet.txt (100%)

$ echo "123456789" >> alphabet.txt
$ cat alphabet.txt
adcdefghijklmnopqrstuvwxyz
123456789
$ git checkout alphabet.txt
$ cat alphabet.txt
adcdefghijklmnopqrstuvwxyz
```



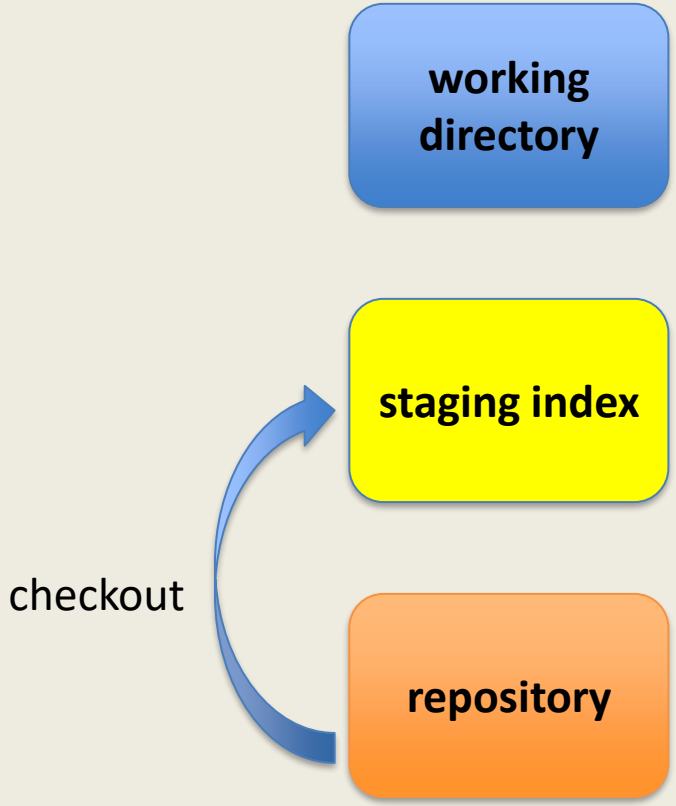
Undo changes made to the staging

```
$ git reset HEAD <file>

$ echo "123456789" >> alphabet.txt
$ git add alphabet.txt
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   alphabet.txt
#
$ git reset HEAD alphabet.txt
Unstaged changes after reset:
M       alphabet.txt
$ git status
# On branch master
# 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:   alphabet.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```



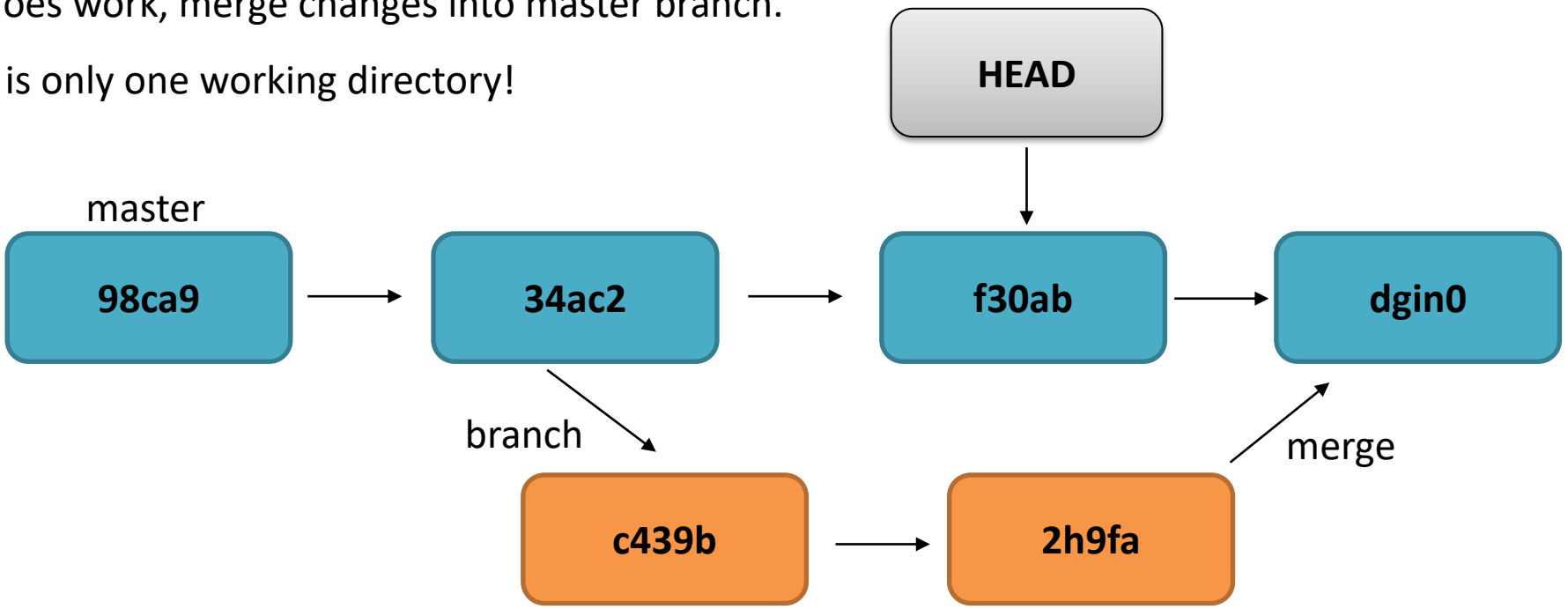
Reverting to older versions

```
$ git checkout <commit> --<file>

$ git commit -a -m "added alphabet for revert demo"
[master 331b9b2] added alphabet for revert demo
 1 file changed, 1 insertion(+)
$ git log -2
commit 331b9b2ab3c8f19c4b351b59d9990843c5770778
Author: s191529 <daniela.daniel@utsouthwestern.edu>
Date:   Mon Dec 7 17:32:20 2020 -0600

    added alphabet for revert demo

commit ab9bffd9a94b76b6c9050526317c5d7c6e9f490a
Author: s191529 <daniela.daniel@utsouthwestern.edu>
Date:   Mon Dec 7 16:43:14 2020 -0600

    renamed foo.txt
$ git checkout ab9bffd9 -- alphabet.txt
$ cat alphabet.txt
adcdefghijklmnopqrstuvwxyz
```

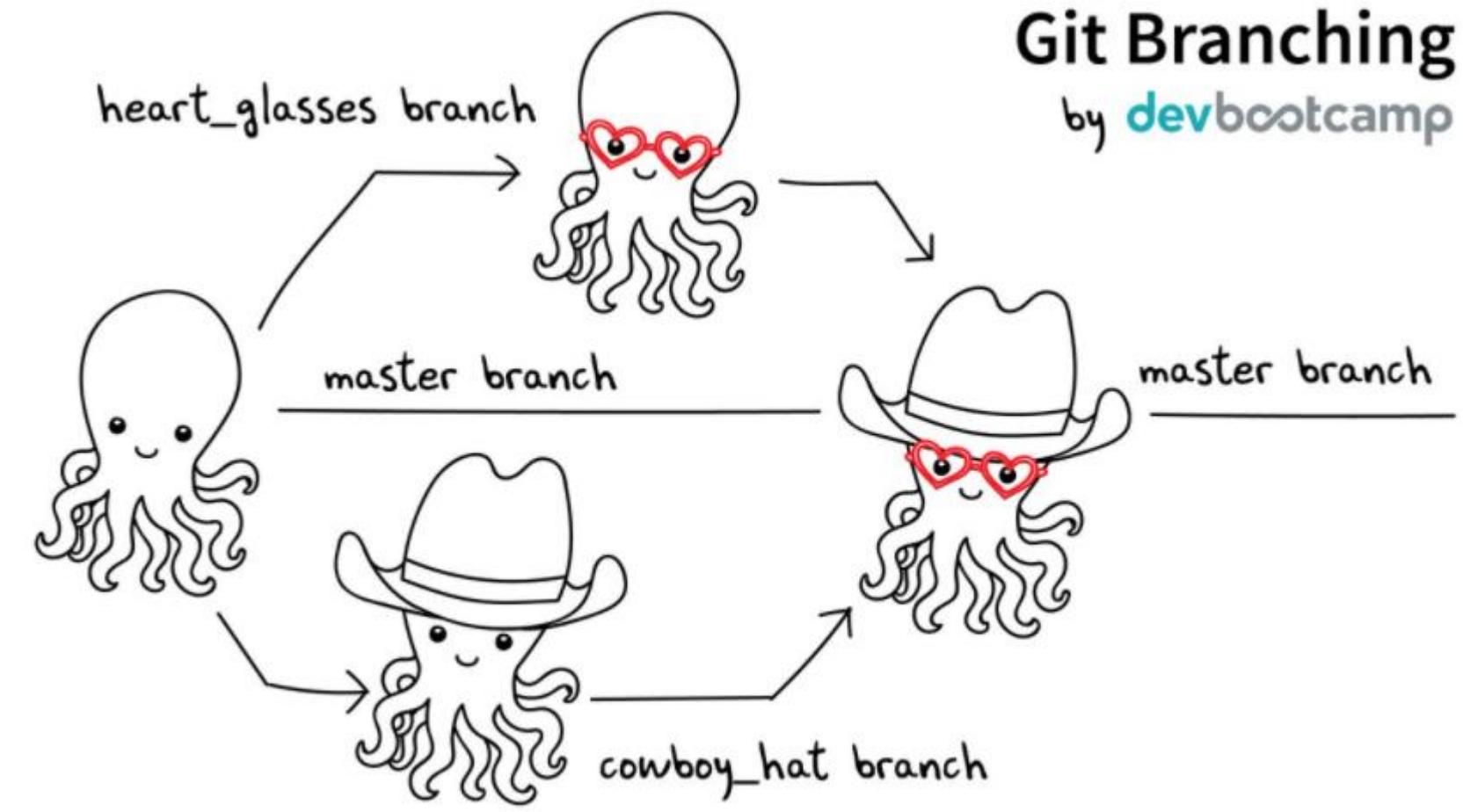


Branching

- Allows one to try new ideas.
- If an idea does not work, throw away the branch. One does not have to undo many changes to master branch.
- If the idea does work, merge changes into master branch.
- Note: there is only one working directory!



Branching is your friend



Source: https://twitter.com/jay_gee/status/703360688618536960

Creating a branch

`git branch` – lists branches and displays current branch with **`*`**

```
$ git branch
* master
```

`git checkout -b <branch>` – creates a new branch from HEAD

```
$ git checkout -b newFeature
M       alphabet.txt
Switched to a new branch 'newFeature'
$ git branch
 master
* newFeature
```

`git checkout <branch>` – change to existing branch

Commits can be made independently to each branch.

Comparing branches

```
git diff master..<branch>
$ echo "ABCDEFGHIJKLMNOPQRSTUVWXYZ" >> alphabet.txt
$ git commit -a -m "capitalized alphabet"
[newFeature 12822c2] capitalized alphabet
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git diff master..newFeature
diff --git a/alphabet.txt b/alphabet.txt
index ae9d20c..80589c5 100644
--- a/alphabet.txt
+++ b/alphabet.txt
@@ -1,2 +1,2 @@
 adcdefghijklmnopqrstuvwxyz
-123456789
+ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

How do I merge a branch?

git merge <branch> – merges branch into current branch

```
$ git checkout master
Switched to branch 'master'
$ git status
# On branch master
nothing to commit, working directory clean
$ cat alphabet.txt
adcdefghijklmnopqrstuvwxyz
123456789
$ git merge newFeature
Updating 331b9b2..12822c2
Fast-forward
 alphabet.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
$ cat alphabet.txt
adcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Merge conflicts!

Merge conflicts are a normal experience of a VCS:

- When two branches have the same file with different content.
- If a file was removed while a branch had it modified.

Git will often resolve conflicts automatically, but in some cases a developer needs to resolve these manually.

```
> git merge issue53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit
the result.
```

Creating a simple merge conflicts

Create **foo.txt** on **master** branch*:

```
$ vi foo.txt
This is to test merge conflicts!!!
$ git add foo.txt
$ git commit -m "foo.txt on master branch" foo.txt
$ git checkout newFeature
```

Create **foo.txt** on **newFeature** branch:

```
$ vi foo.txt
This is my other 'foo.txt' file...
$ git add foo.txt
$ git commit -m "foo.txt on newFeature branch" foo.txt
$ git merge master
Auto-merging foo.txt
CONFLICT (add/add): Merge conflict in foo.txt
Automatic merge failed; fix conflicts and then commit the
result.
```

*File **alphabet.txt** was removed from the local repo.

Dealing with merge conflicts

`git merge --abort` and resolve conflict manually.

Then attempt to merge again. Tips to reduce the pain of merge conflicts:

- Merge often;
- Keep commits small/focused;
- Bring changes occurring to master into your branch frequently (“tracking”).

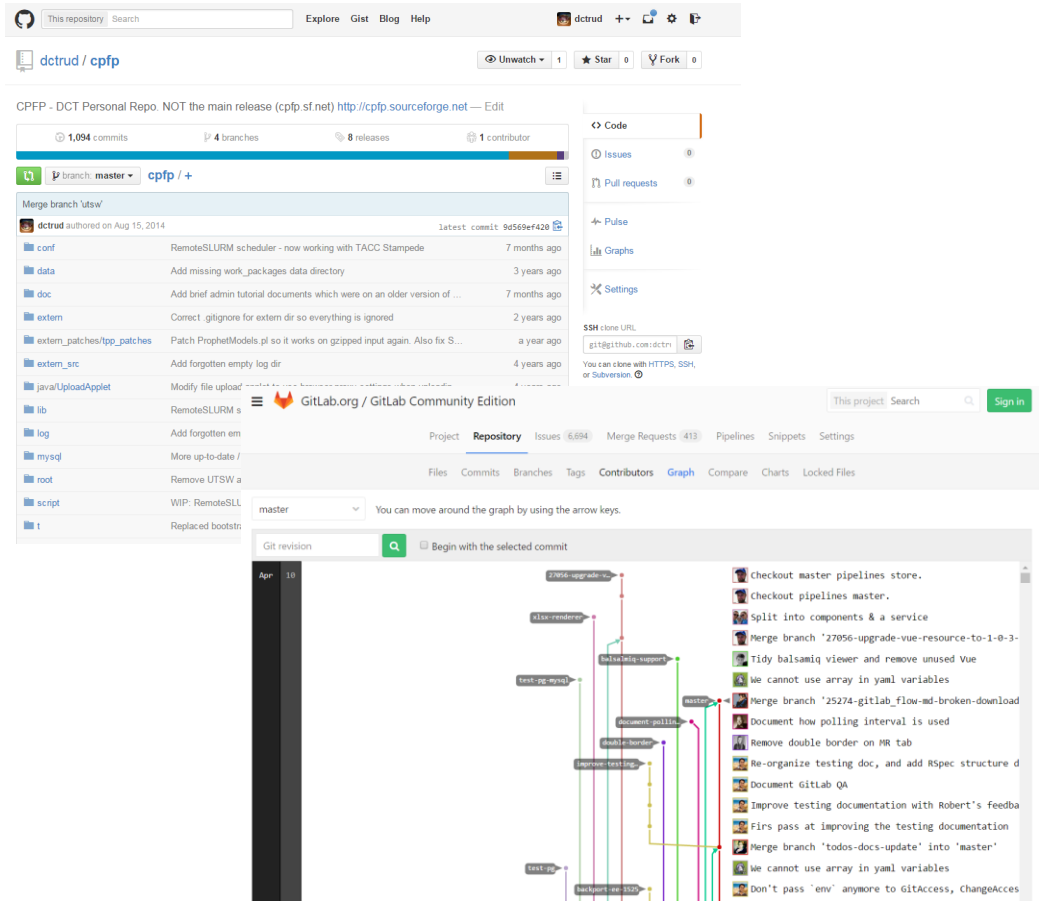
```
$ cat foo.txt
<<<<<<< HEAD
This is my other 'foo.txt’ file...
=======
This is to test merge conflicts!!!
>>>>>>> master
$ git merge --abort
$ cat foo.txt
This is my other 'foo.txt’ file...
```

Online (remote) repository hosting

Most used site by far!

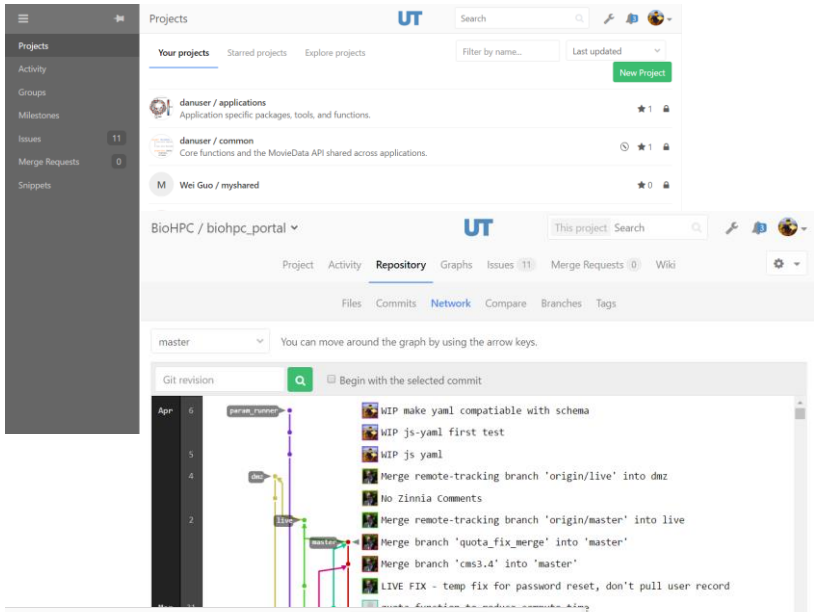
GitHub

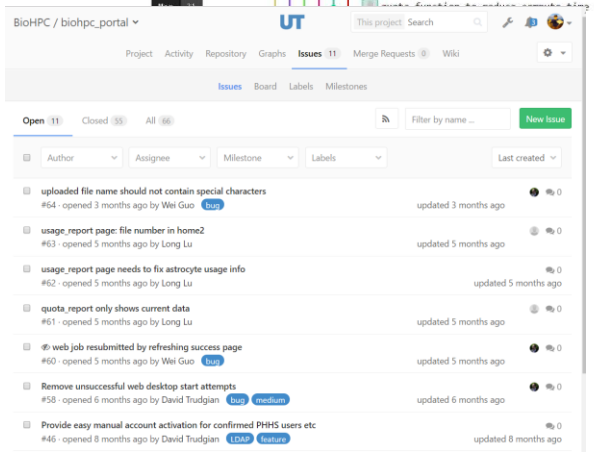
GitLab



We have https://git.biohpc.swmed.edu – local GitLab, it's a lot like GitHub.

git.biohpc.swmed.edu





Browse files and history

Manage access rights

Create branches/forks

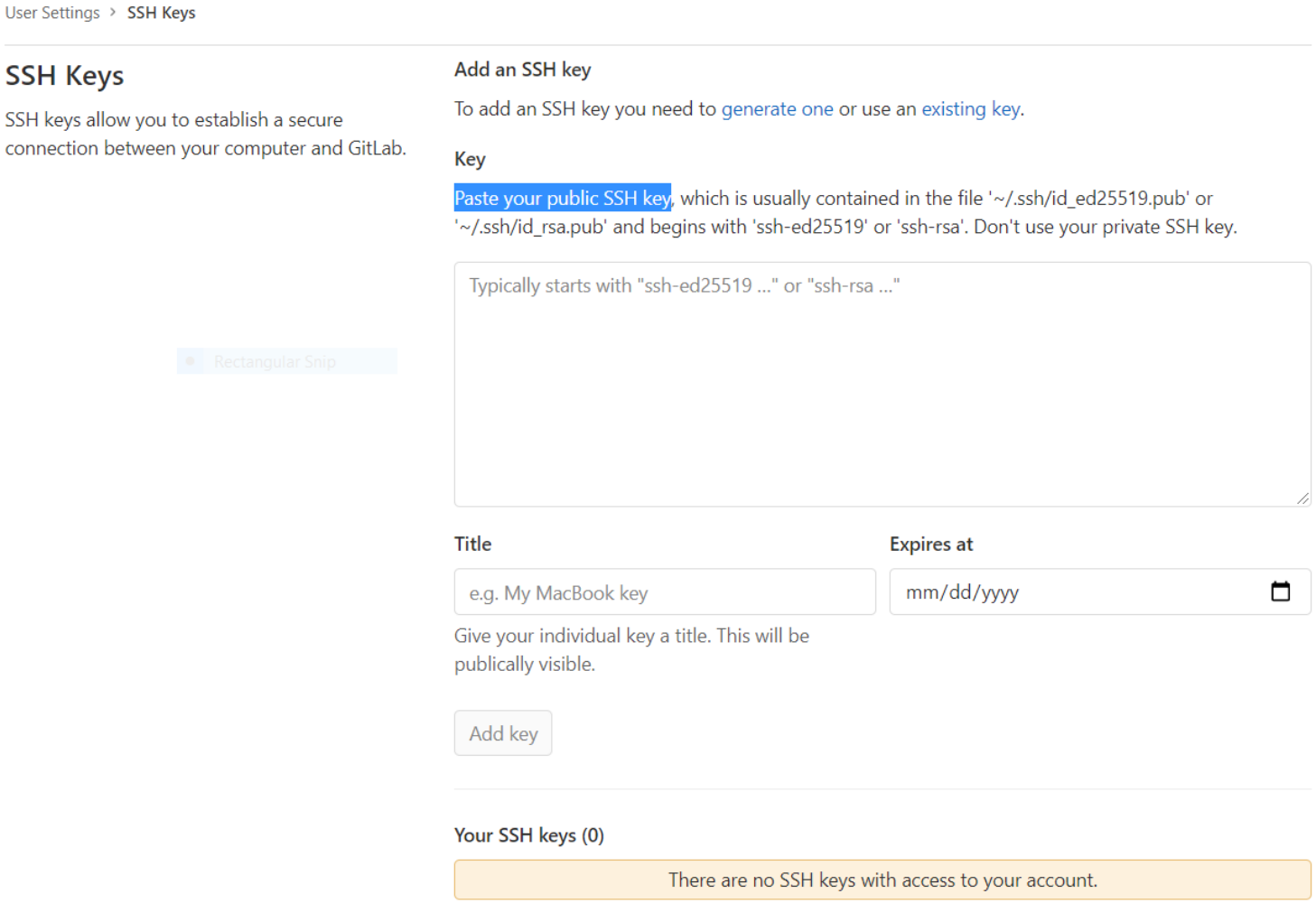
Perform *basic* editing online

Track issues/bugs, merge requests

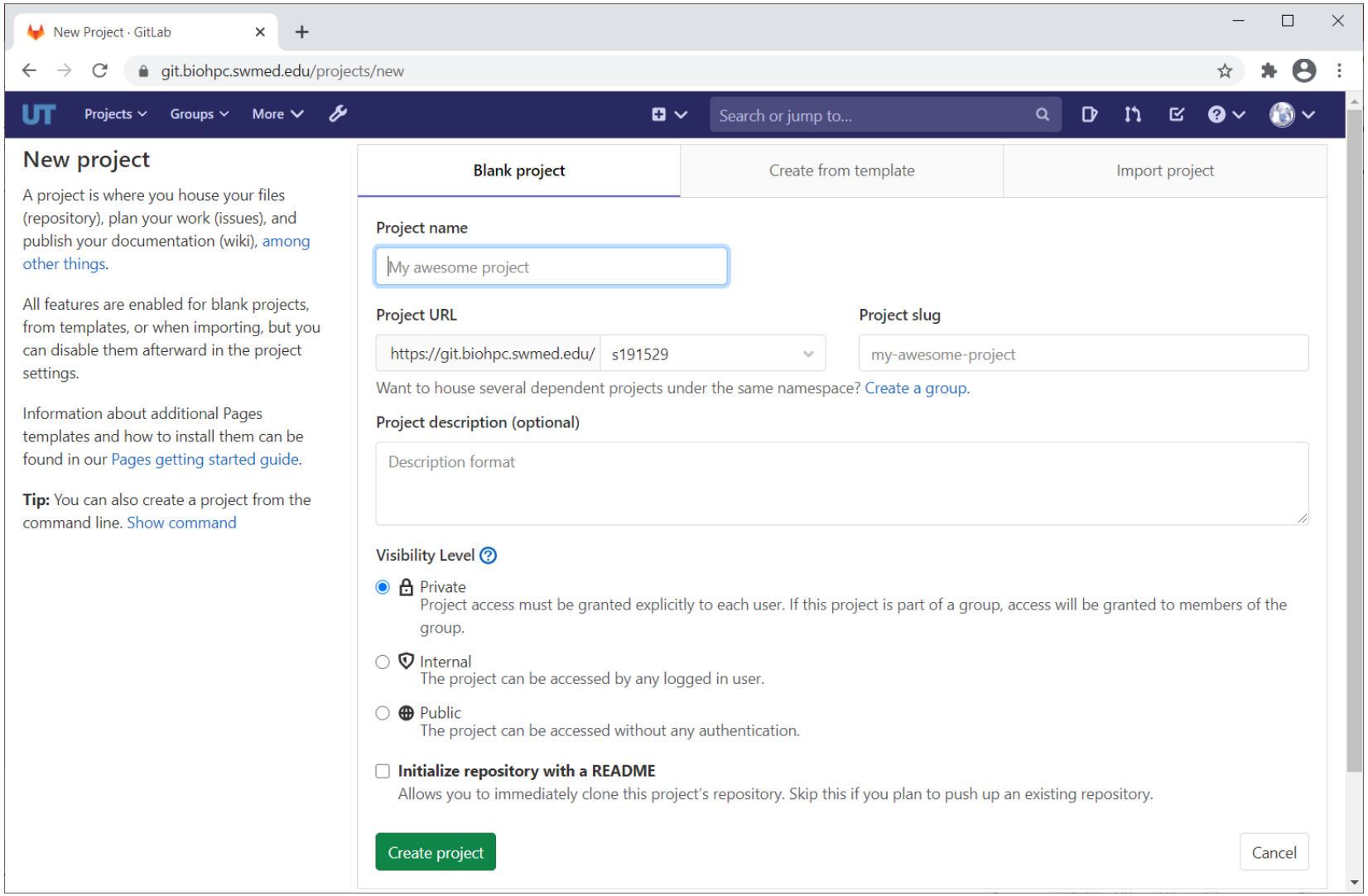
Accessible from the internet

Accounts for non-UTSW collaborators available*

Set up SSH credentials on *git.biohpc.swmed.edu*



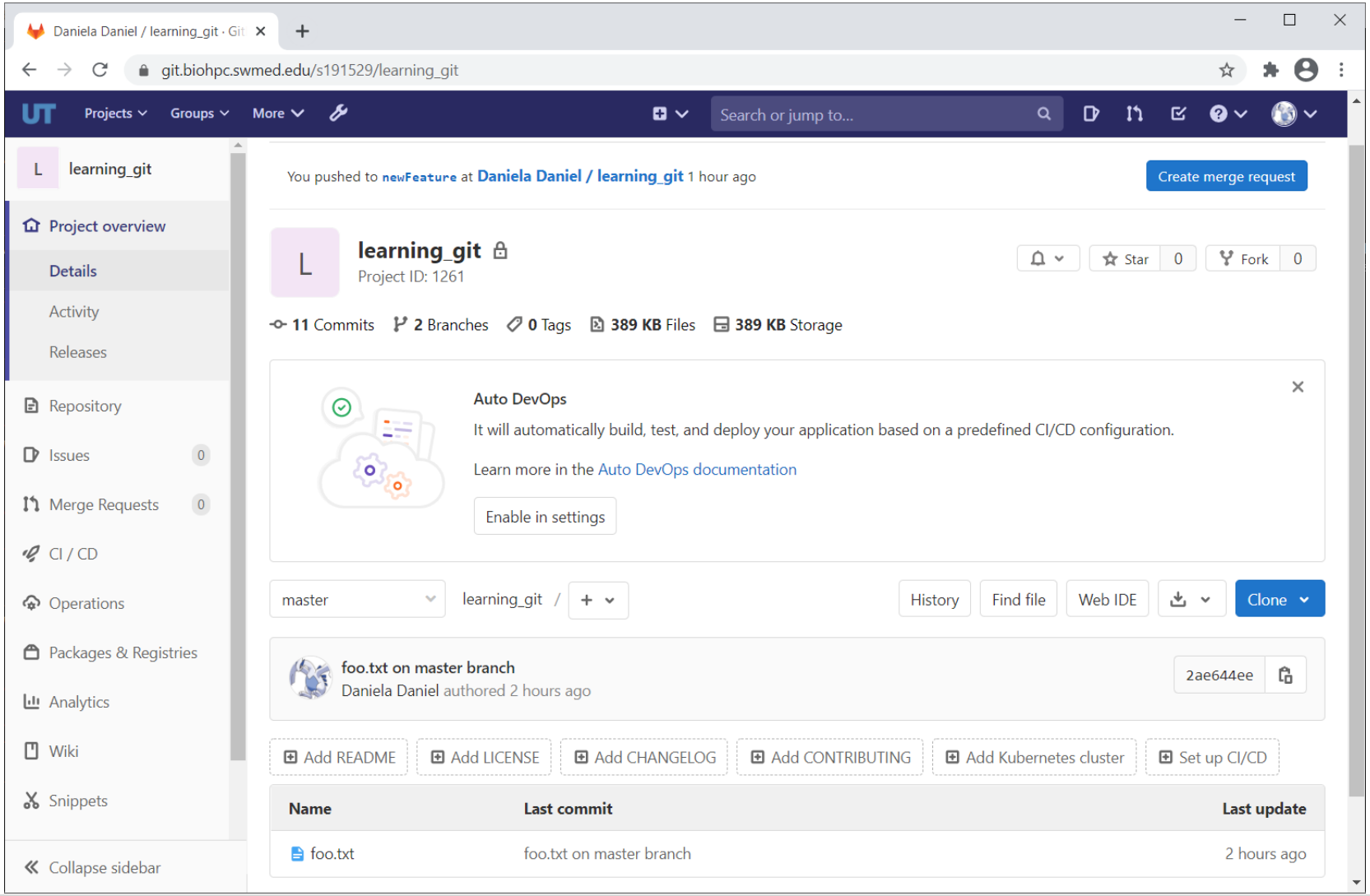
Creating a new project on GitLab



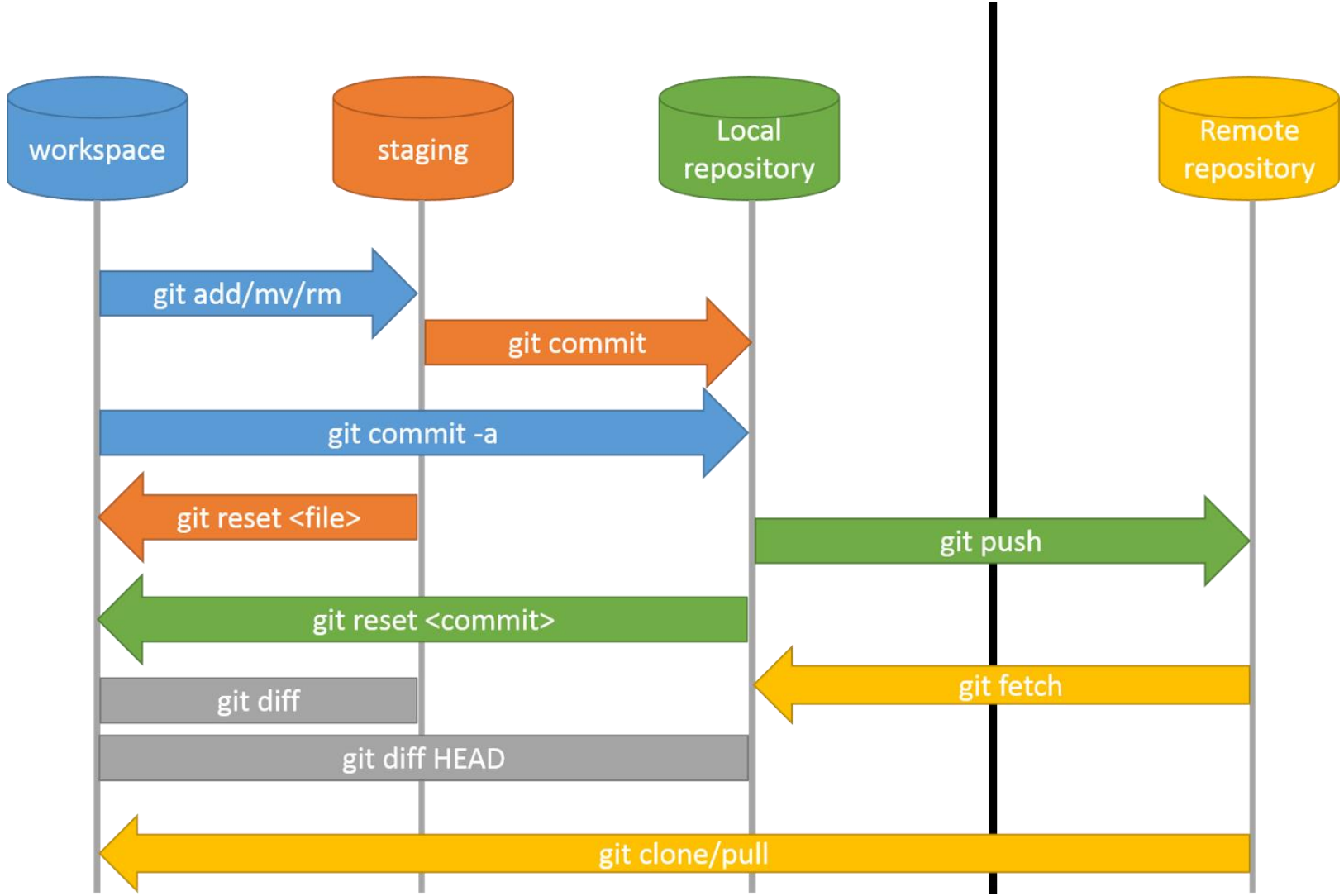
Push Existing Repo to New Project

```
$ git remote add origin git@git.biohpc.swmed.edu:s191529/learning_git.git

$ git remote -v
origin  git@git.biohpc.swmed.edu:s191529/learning_git.git (fetch)
origin  git@git.biohpc.swmed.edu:s191529/learning_git.git (push)

$ git push -u origin --all
Counting objects: 31, done.
Delta compression using up to 32 threads.
Compressing objects: 100% (16/16), done.
Writing objects: 100% (31/31), 2.65 KiB | 0 bytes/s, done.
Total 31 (delta 2), reused 0 (delta 0)
remote: To create a merge request for newFeature, visit:
remote:   https://git.biohpc.swmed.edu/s191529/learning_git/-
/merge_requests/new?merge_request%5Bsource_branch%5D=newFeature
remote: The private project s191529/learning_git was successfully created.
remote: To configure the remote, run:
remote:   git remote add origin git@git.biohpc.swmed.edu:s191529/learning_git.git
remote: To view the project, visit:
remote:   https://git.biohpc.swmed.edu/s191529/learning_git
To git@git.biohpc.swmed.edu:s191529/learning_git.git
 * [new branch]      master -> master
 * [new branch]      newFeature -> newFeature
Branch master set up to track remote branch master from origin.
Branch newFeature set up to track remote branch newFeature from origin.
```

This is how the project looks on GitLab



Push and pull to/from the remote repo



Summary

`git checkout -b <branch>` (create and switch to new branch)

`git branch -d <branch>` (delete a branch)

`git merge <branch>` (merge with another branch)

`git diff <source> <destination>` (preview before merging branches)

`git log` (study the log)

`git log --graph --oneline --decorate --all` (fancy)

`git checkout -- <filename>` (replace local changes)

`git fetch origin` (drop local changes)

`git reset --hard origin/master` (reset)

Edit the file `.gitignore`

`git config --list` (list configuration values)

Resources

Notes Specific to the BioHPC git server
https://portal.biohpc.swmed.edu/content/guides/using-biohpc-git/

Tutorials
https://www.atlassian.com/git/tutorials/

Videos
https://www.youtube.com/watch?v=r63f51ce84A