

---

# Introduction to Git and GitLab

[web] [portal.biohpc.swmed.edu](http://portal.biohpc.swmed.edu)

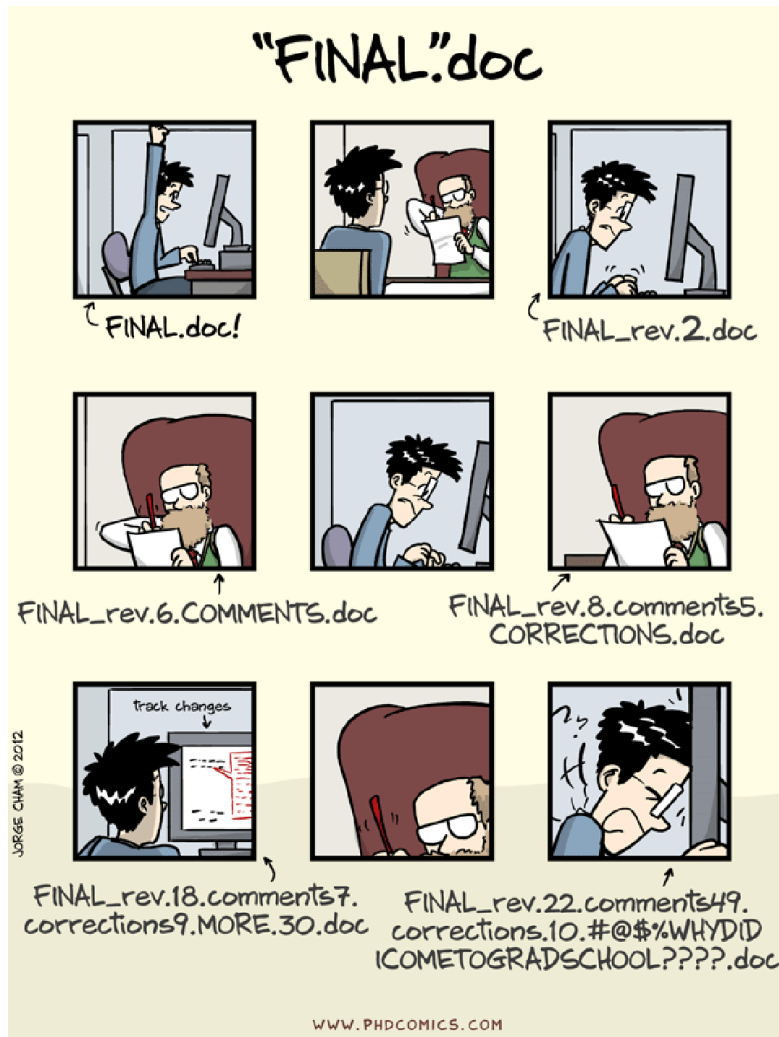
[email] [biohpc-help@utsouthwestern.edu](mailto:biohpc-help@utsouthwestern.edu)

## Agenda

---

- Why do we need Git?
- What *is* Git?
  
- Git Basics – using it locally
- Git Basics – using remotes
  
- GitLab
  - SSH keys
- GitLab Projects

## Why do we need version control systems (VCS-es)?



- Organize changes and revisions in a meaningful way!
- Large codebases, scientific software...
  - Many collaborators
  - Many versions, bugfixes...
- A VCS is *not* a backup system.
  - Often not optimized for large files

## What is Git?

- Git is a ***distributed version control system.***
- Invented by Linus Torvalds (the Linux guy) in 2005
- Version Control System – **track changes** in a code-base
- Distributed – no one ‘golden repo’
  - As opposed to centralized (e.g. SVN)



## Git – basic terminology

Git is a **version control system**, designed to **track changes** to your codebase.

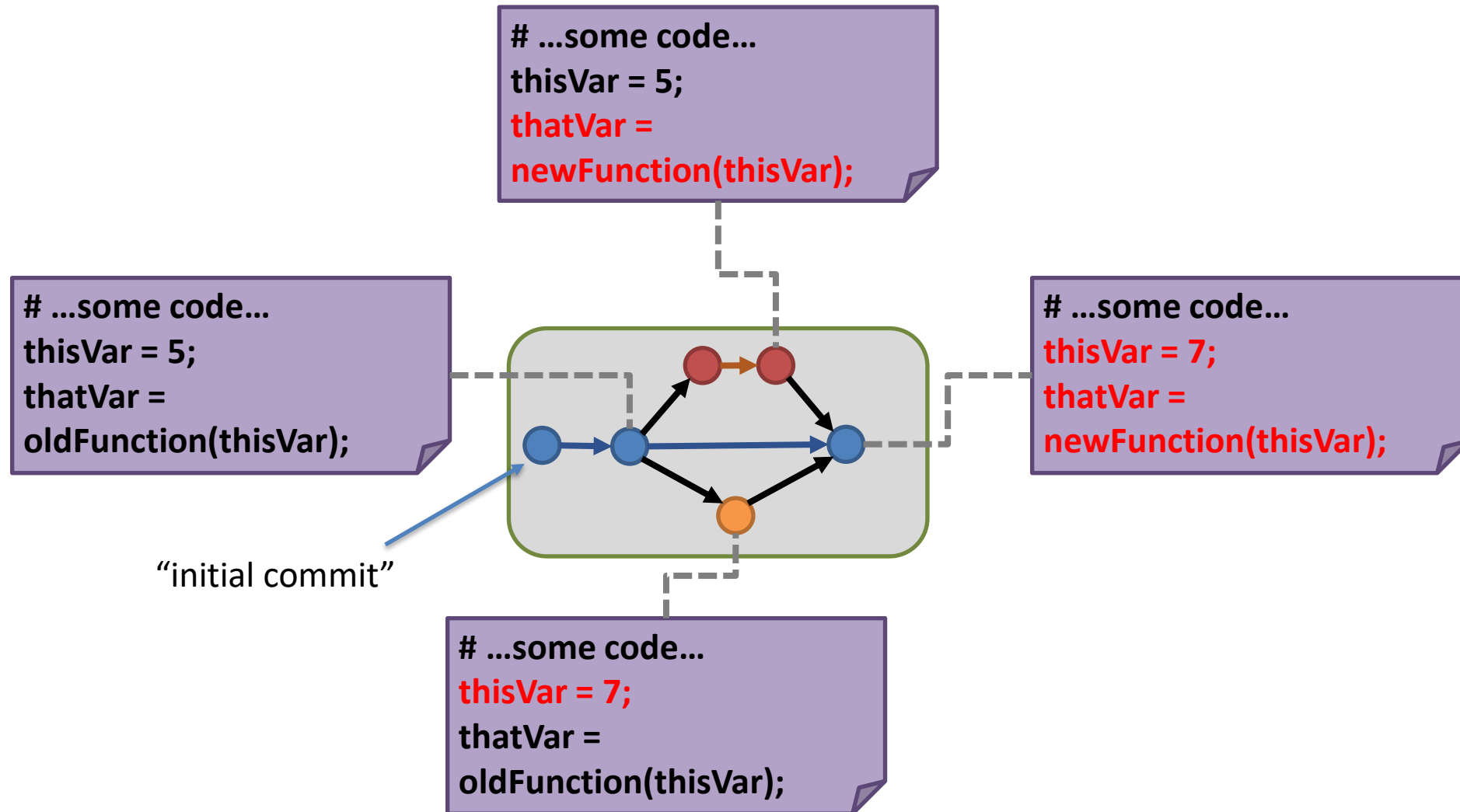
- A **git repository** is a collection of code, tracked by git.
- A **git commit** is a set of changes, applied to some previous repository state, that updates the repository to some new state.
  - A given commit is referred to by its **commit ID**

```
commit ef98cdf4d8976013c9002cf60f79677fac812ee
Author: user_name <User.Name@UTSouthwestern.edu>
Date:   Tue Nov 8 09:27:55 2022 -0600

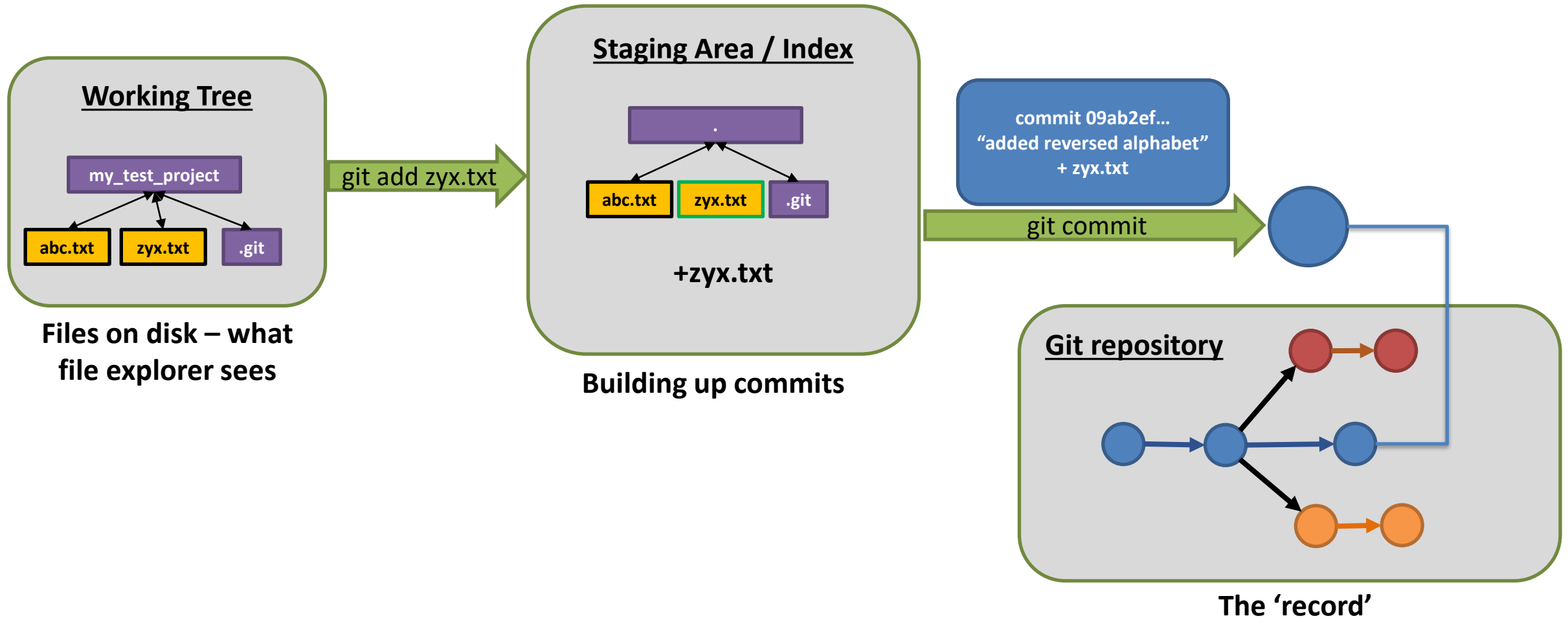
    New alphabet reference file
```

- A **git push** is an action that migrates those changes to some other repository (e.g. Gitlab)
- A **git branch** is a series of related commits distinct from other branches.
- A **git merge** is a process of bringing changes from one branch to your current one.
- A **git tag** is a ‘special name’ given to a particular commit – stored in the **ref-log**

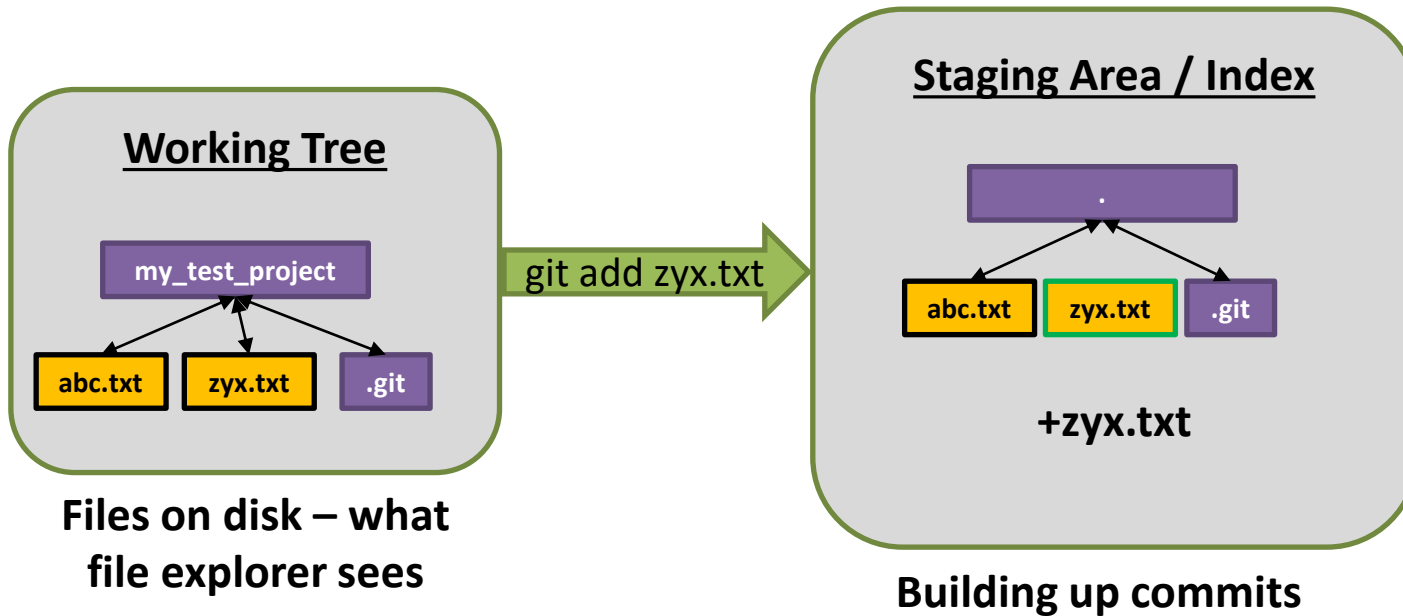
## Git as a graph of 'repository states'



## The three areas of your local Git environment



## The three areas of your local Git environment – adding to the index



When you save a file, you are saving it to your filesystem. Git sees this as the 'working tree'.

The Staging Area is where you build up a commit – you can add and remove files, eventually building up a 'change unit' called a commit.

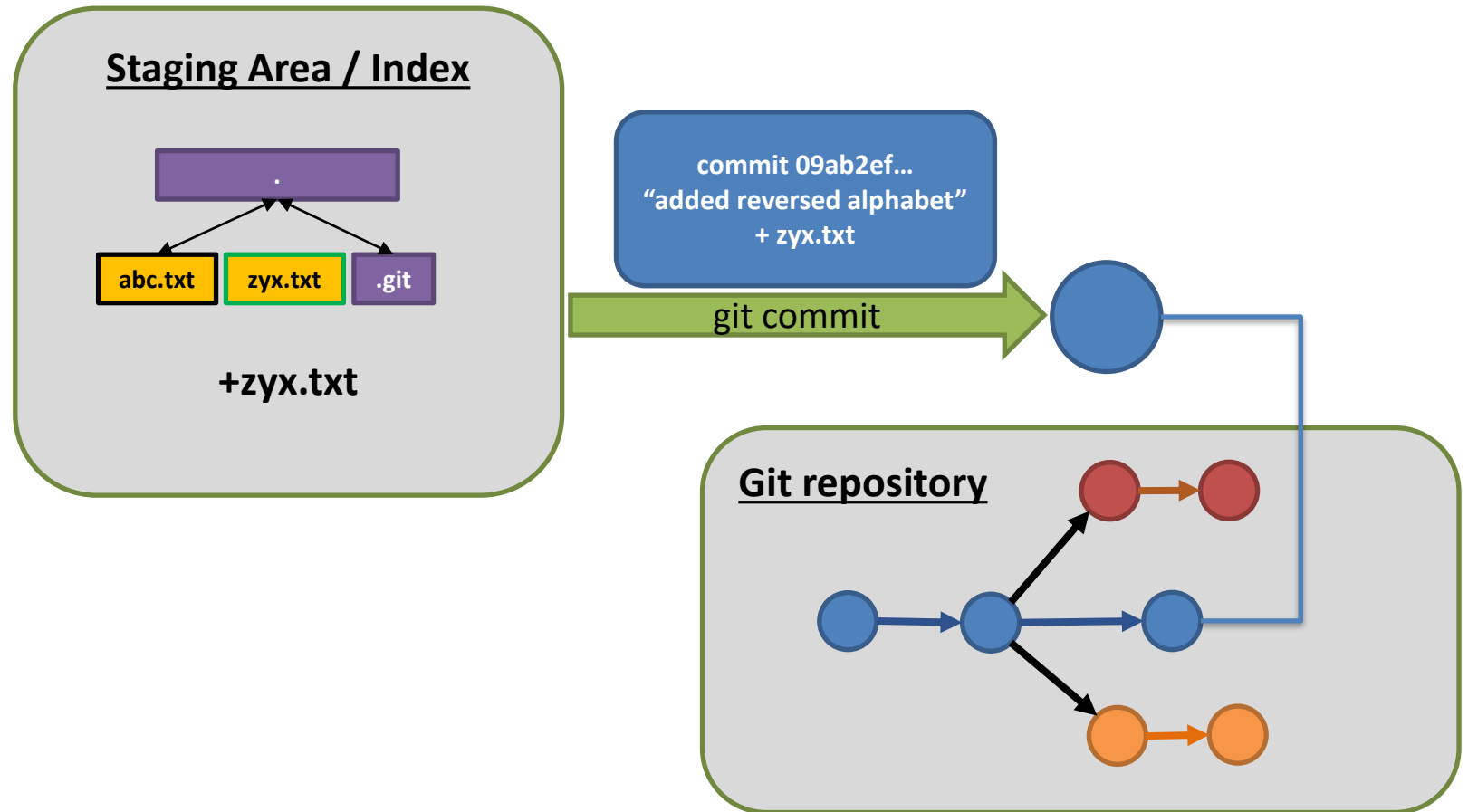


## The three areas of your local Git environment – Committing your commits

**git commit** ‘seals and stamps’ your staged changes, entering them (along with a commit message) to the repository. At this point, the commit is in the record.

- **commit ID** entered in the **reflog**

You can now build up another commit in the staging area (e.g. a different function you’ve implemented), or sync with an external repository.



## git config

---

- Before using git, you may want to set a few configuration variables.
  - git may prompt you to do this on your first push.
- Most if not all have both **local** and **global** options
  - **local** is for the currently active repo
  - **global** is system-wide

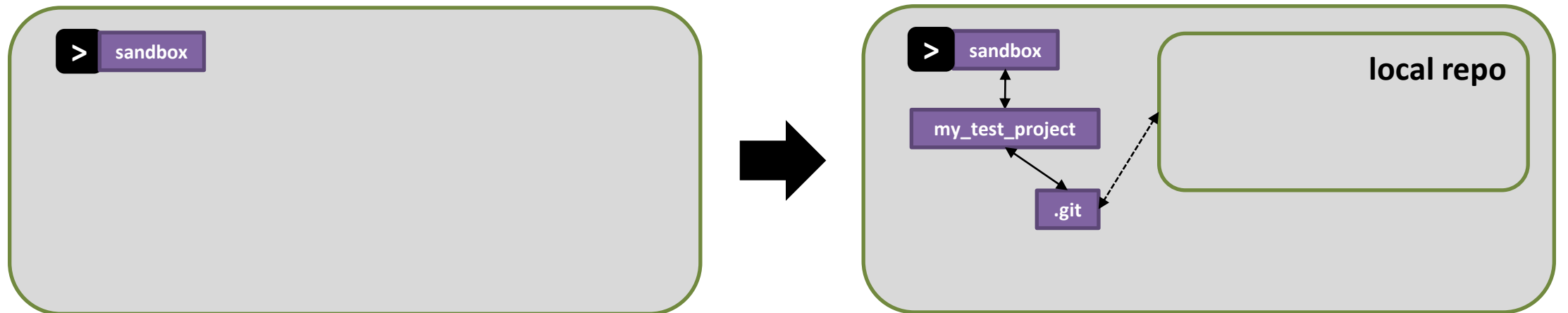
```
$ git config --global user.name your_username
$ git config --global user.email your.email@utsouthwestern.edu

$ git config --local user.email your.other.email@gmail.com
```

- There are many options – you might have to use a few depending on your needs.

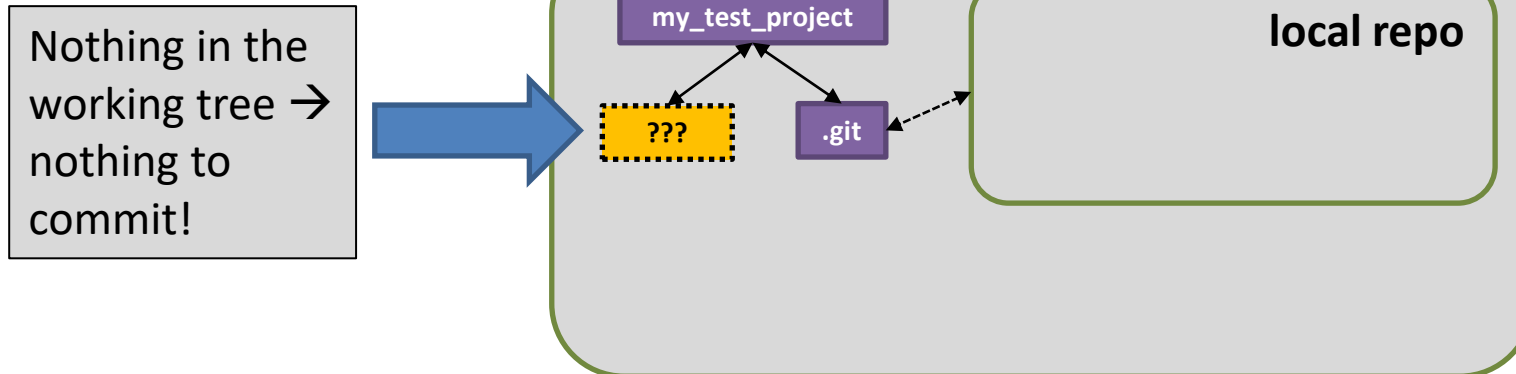
## git init – creating a repository

```
my_user@my_host:sandbox$ git init my_test_project
Initialized empty Git repository in /home/sandbox/my_test_project/.git/
my_user@my_host:sandbox$ ls
my_test_project
my_user@my_host:sandbox$ ls my_test_project/ -a
.  ..  .git
```



## git status – checking on your git workspace

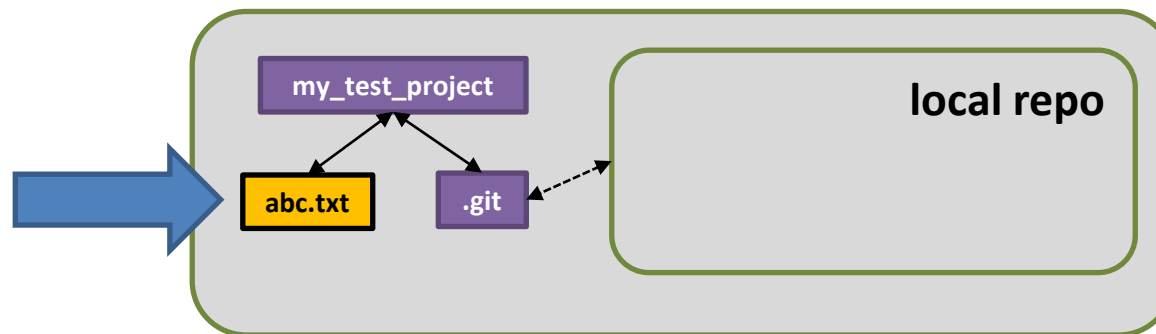
```
my_user@my_host:my_test_project (master)$ git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```



## git status – Git only tracks what you've told it to.

```
my_user@my_host:my_test_project (master)$ echo "abcdefghijklmnopqrstuvwyz" > abc.txt
my_user@my_host:my_test_project (master)$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       abc.txt
nothing added to commit but untracked files present (use "git add" to track)
```

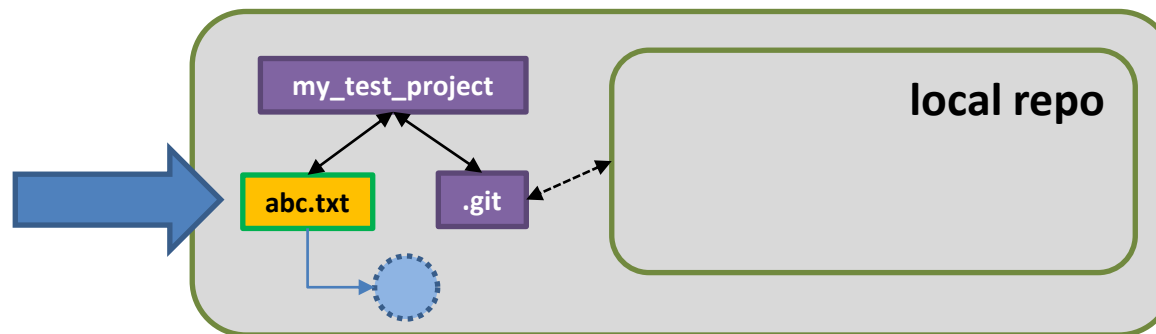
Files not already  
added to git repository  
= untracked



## git add – telling Git what to track

```
my_user@my_host:my_test_project (master)$ git add abc.txt
my_user@my_host:my_test_project (master)$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   abc.txt
#
```

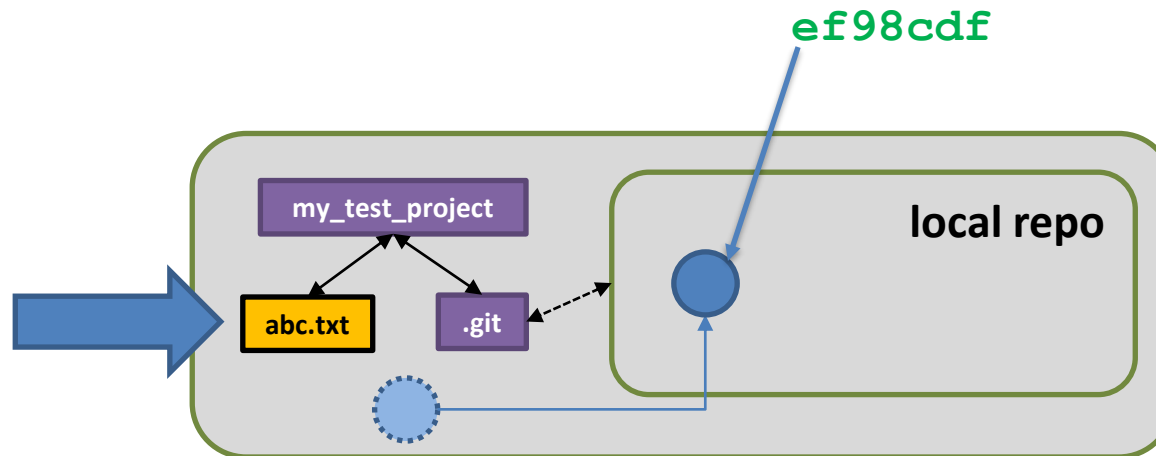
git add makes the untracked file part of the next commit



## git commit – entering changes into the repository

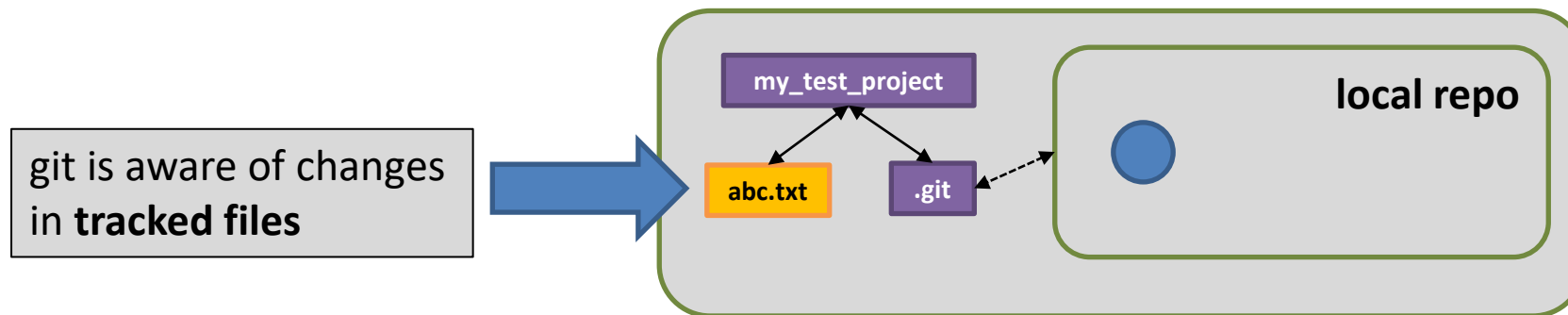
```
my_user@my_host:my_test_project (master)$ git commit -m "New alphabet reference
file"
[master (root-commit) ef98cdf] New alphabet reference file
1 file changed, 1 insertion(+)
create mode 100644 abc.txt
my_user@my_host:my_test_project (master)$ git status
# On branch master
nothing to commit, working directory clean
```

git commit 'zips up'  
the commits, tacks on  
the message,  
generates ID, and  
updates local repo.



## git status – Git naturally looks at differences.

```
my_user@my_host:my_test_project (master)$ echo "abcdefghijklmnopqrstuvwxyz" >
abc.txt
my_user@my_host:my_test_project (master)$ 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:   abc.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```





## git diff – comparing different files from different commits

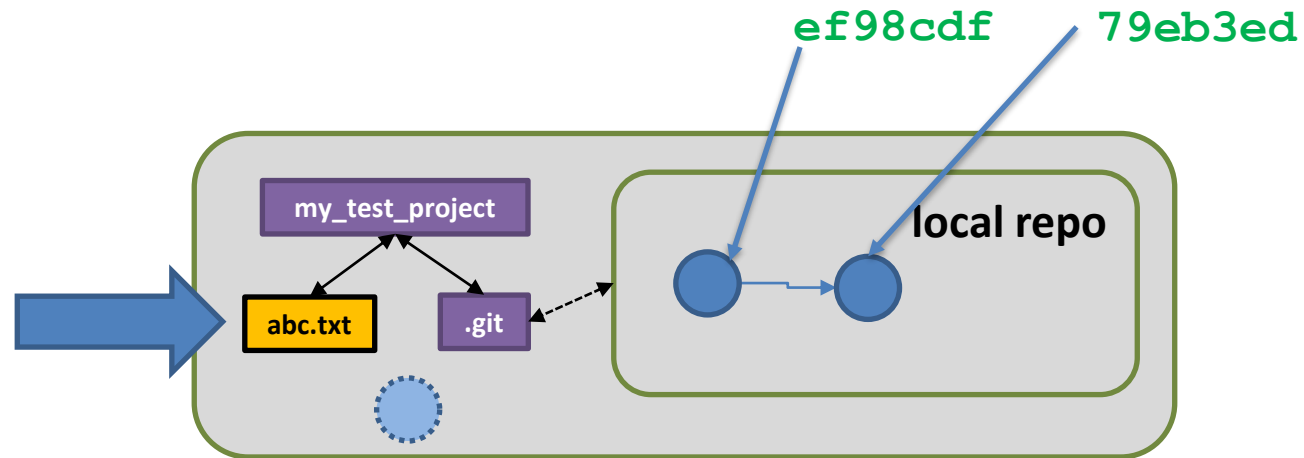
```
git diff <commit> <commit>
```

```
git diff alone will compare the working directory with HEAD
```

```
git diff -staged will compare the staging area with HEAD
```

```
git diff <filename> will restrict comparison to a specific file.
```

git commit 'zips up' the commits, tacks on the message, generates ID, and updates local repo.



## git diff will compare LINE-BY-LINE

```
$ git diff
diff --git a/abc.txt b/abc.txt
index 0ce81d4..b0883f3 100644
--- a/abc.txt
+++ b/abc.txt
@@ -1,1 @@
-abcdefghijklmnopqrstuvwyz
+abcdefghijklmnopqrstvwxyz

$ git diff abc.txt
diff --git a/abc.txt b/abc.txt
index 0ce81d4..b0883f3 100644
--- a/abc.txt
+++ b/abc.txt
@@ -1,1 @@
-abcdefghijklmnopqrstuvwyz
+abcdefghijklmnopqrstvwxyz
```

## Moving and renaming files – why not to use mv

```
$ mv abc.txt xyz.txt
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add/rm <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       deleted:    abc.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       xyz.txt
no changes added to commit (use "git add" and/or "git commit -a")
$ mv xyz.txt abc.txt
$ git status
# On branch master
nothing to commit, working directory clean
```

## Moving and renaming files – why to use git mv

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

## Git looking at differences – the commit log

```
$ git add abc.txt
$ git commit -m "Added x"
[master 79eb3ed] Added x
 1 file changed, 1 insertion(+), 1 deletion(-)
my_user@my_host:my_test_project (master)$ git log
commit 79eb3ed033f6239ffb41c25e389c5000655a5463
Author: user_name <User.Name@UTSouthwestern.edu>
Date:   Tue Nov 8 09:34:26 2022 -0600

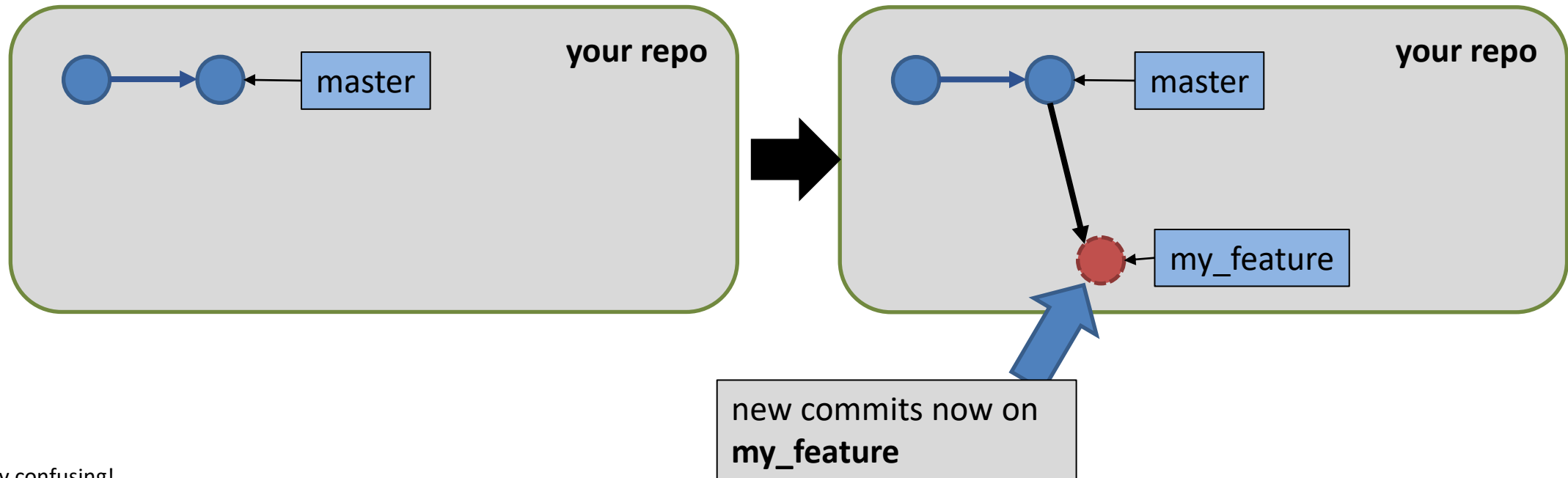
    Added x

commit ef98cdfe4d8976013c9002cf60f79677fac812ee
Author: user_name <User.Name@UTSouthwestern.edu>
Date:   Tue Nov 8 09:27:55 2022 -0600

    New alphabet reference file
```

## Branching, commits, and HEAD

- HEAD is shorthand for ‘the commit in the repository from which your current working tree was derived’
  - Points to the **last commit that you made, or to the last commit that was checked out.**
- Branching creates a new named ref – the next commit will be on this branch!



This can be very confusing!  
Check out <https://jwiegley.github.io/git-from-the-bottom-up> for a good explanation

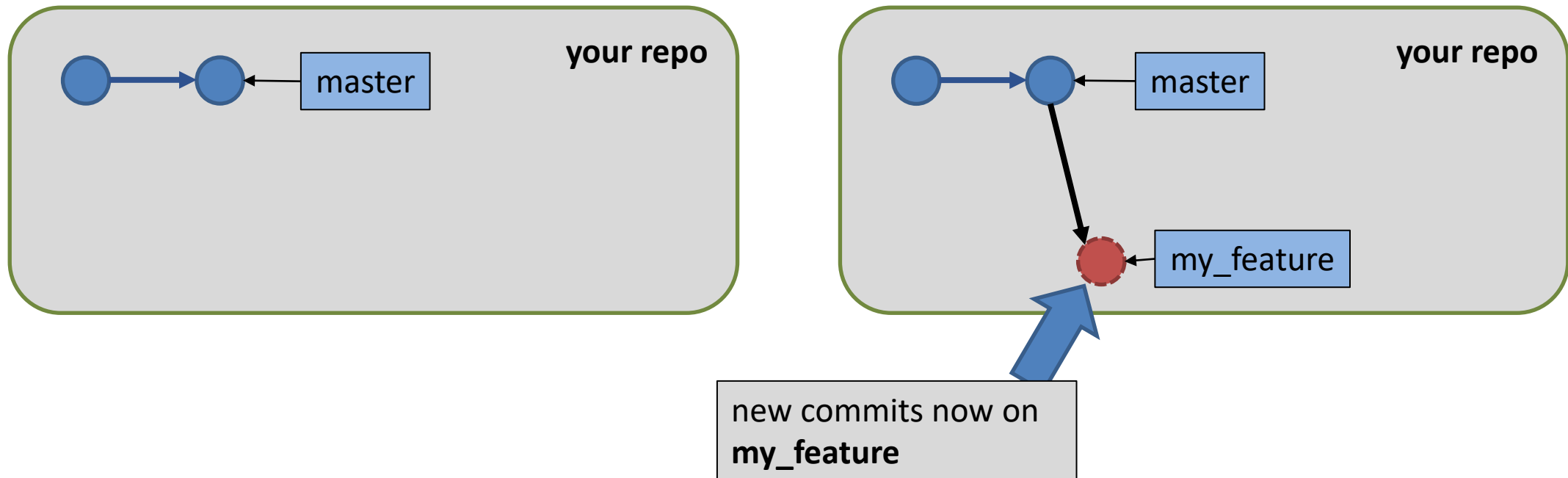
## git branch – creating parallel timelines

- Branches are **the** most important part of having a well-organized codebase that many people can work on.

```
$ git branch my_feature && git checkout my_feature
```

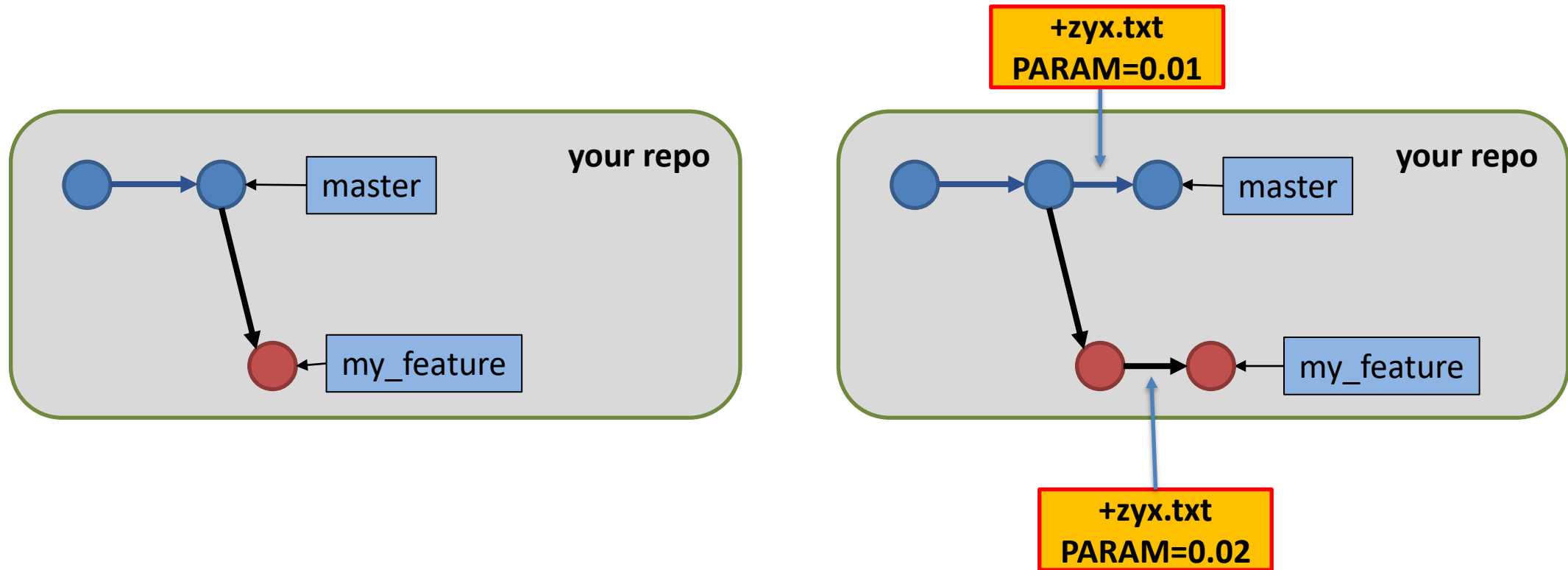
OR

```
$ git checkout -b my_feature
```



## Branches allow parallel workflows

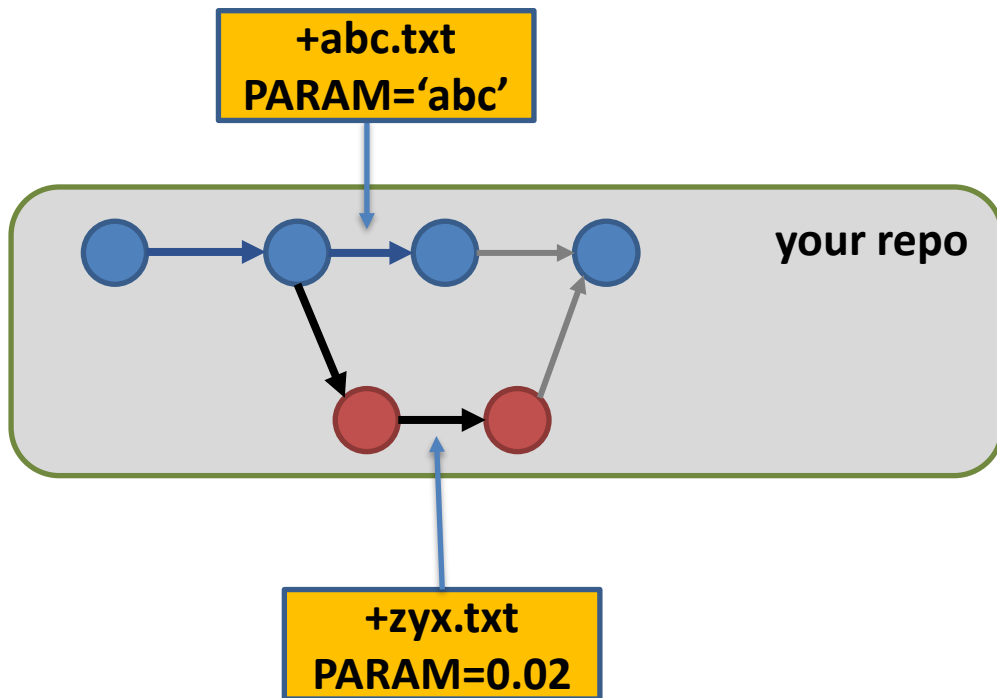
- Separate branches → Separate features
  - Must be careful to keep things independent, otherwise you run into the dreaded **merge conflict!**





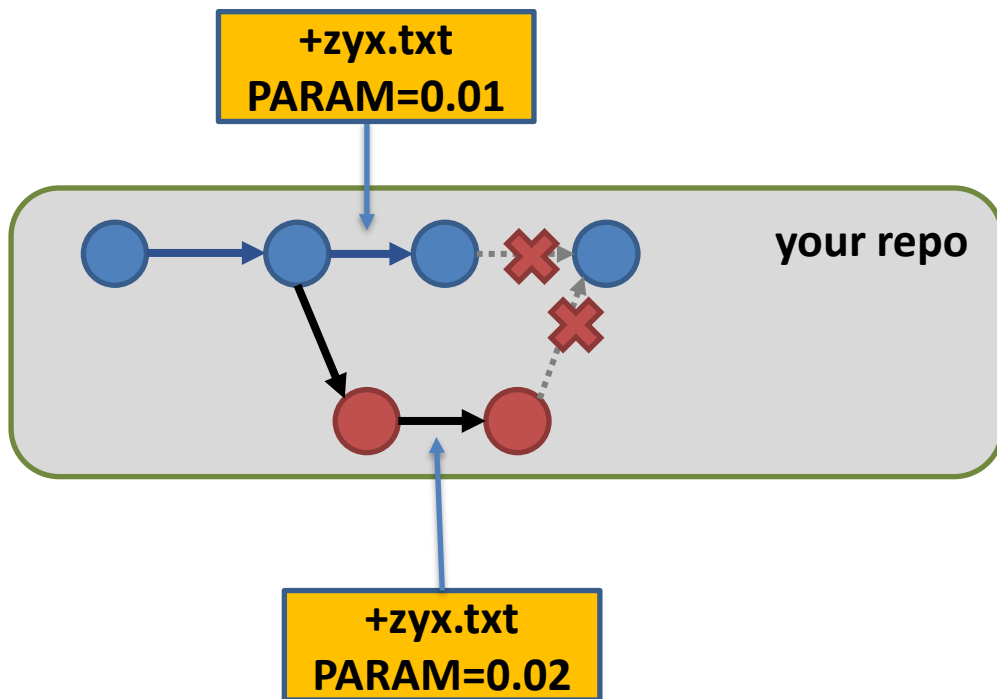
## Merging branches 'finishes' one branch by bringing its history of changes into another.

- Independent changes (i.e. those which generate no merge conflicts) are automatically combined.
- A special '**merge commit**' is generated for this process.



## Merge conflicts arise because of incompatible changes from different branches.

```
(master) $ git merge my_feature
Auto-merging abc.txt
CONFLICT (content): Merge conflict in abc.txt
Automatic merge failed; fix conflicts and then commit the result.
(master | MERGING) $ cat zyx.txt
```



```
<<<<<<< HEAD
PARAM=0.01
=====
PARAM=0.02
>>>>>> my_feature
```

```
(master | MERGING) $ echo "PARAM=0.015" > zyx.txt
(master | MERGING) $ git add zyx.txt
(master | MERGING) $ git commit -m "Averaged PARAM"
(master) $
```

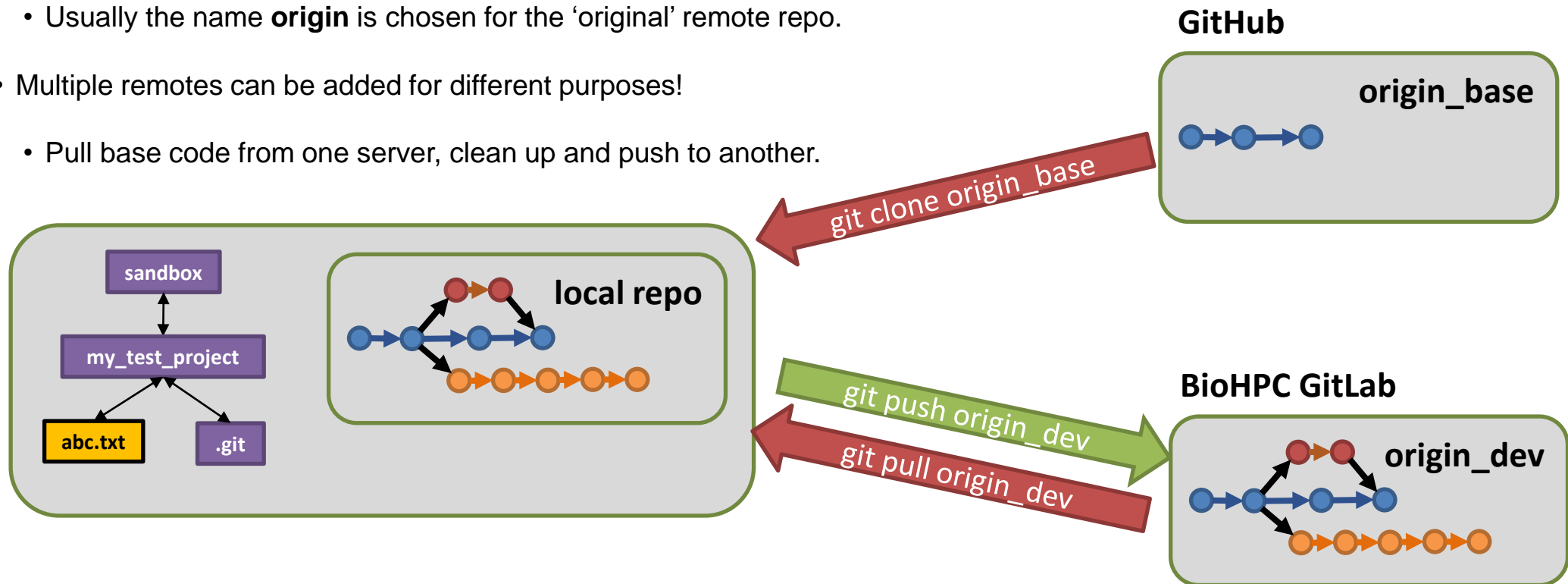
## Recap – Local Git

---

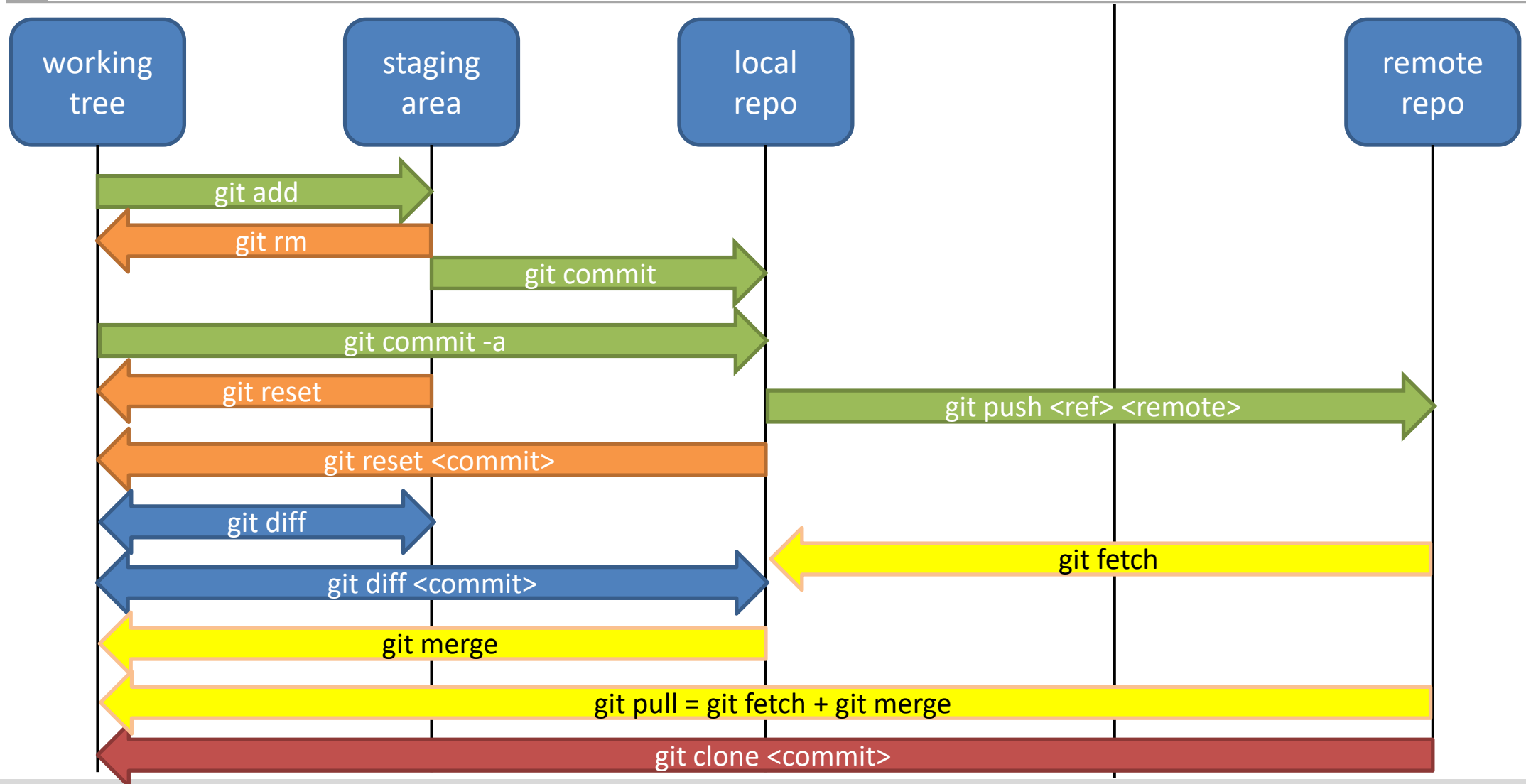
- Three local areas – the **working tree**, the **staging area** or **index**, and the **repository**.
- **Commits** are assembled in the **staging area** and **committed** to the **repository** along with a descriptive message.
  - Each has a checksum which functions as its unique name
  - HEAD refers to the commit that your current working tree is derived from.
- **Branches** are parallel series of **commits** – they are split off with **git branch** and combined with **git merge**.
  - the branch-ref points at the most recent commit in the repository for each branch
  - committing to a branch updates the branch-ref to point at your new commit.
  - **Merge conflicts** arise in cases where Git cannot sensibly combine your changes.
    - Adds another ‘mini commit’ into the process specifically to allow you to manually intervene.

## Using remote repositories

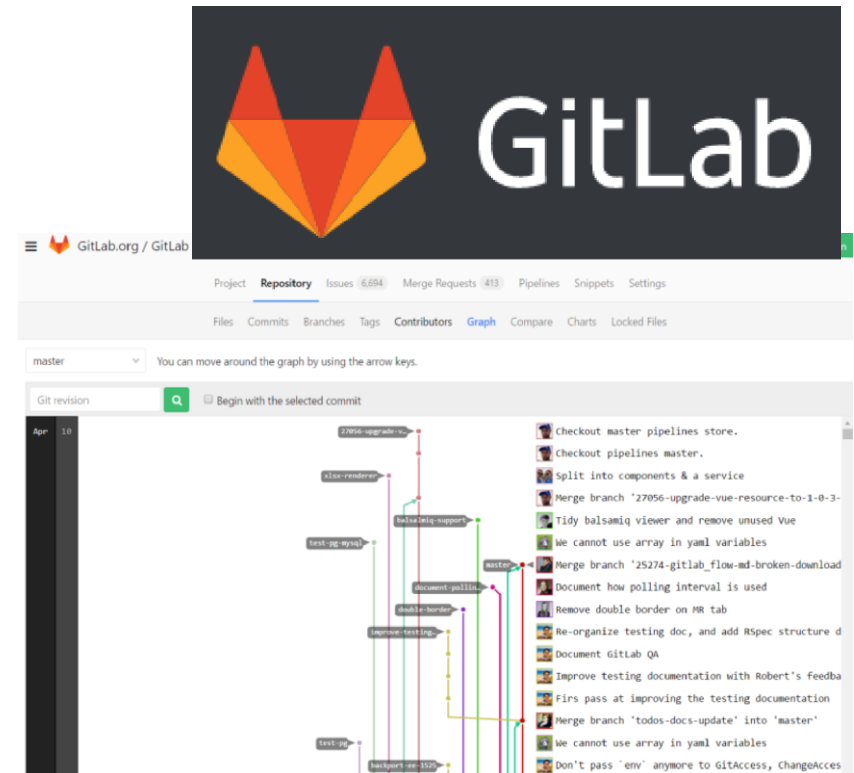
- Git is useful as a version tracking tool for your own purposes, but it shines when used to share code.
- Your local repository can be configured to use multiple remotes
  - Usually the name **origin** is chosen for the 'original' remote repo.
- Multiple remotes can be added for different purposes!
  - Pull base code from one server, clean up and push to another.



## Git – local and remote cheat sheet



- BioHPC-hosted service which provides a repository hosting service, along with:
  - Web IDE
  - Project wikis
    - Markdown, plus additional flavor for GitLab specifically.
  - Group organization
  - CI Runner integration
- Future trainings will focus more on GitLab's features



## SSH Keys

---

- To use GitLab correctly, you must add an SSH **public** key to your profile.
- You CANNOT push via basic password authentication as our GitLab is two-factor enabled.
  - HTTPS repository links will generally not function correctly.
  - HTTPS tokens were made having logged in, so they are already 'authenticated'
- Recommended that you use a GitLab-specific SSH key.
  - Public keys are meant to be shown, but you can never be too careful!

**<https://git.biohpc.swmed.edu/-/profile/keys>**

## Generating a key pair (should generally keep separate key pairs for separate computers)

```
$ ssh-keygen -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/.ssh/id_rsa): /home/fakekeypair
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/fakekeypair.
Your public key has been saved in /home/fakekeypair.pub.
The key fingerprint is:
SHA256:55oP837kOBwUul80K811S2+erwGW5HZEkT8U myusername@my_host
The key's randomart image is:
+---[RSA 4096]---+
|      .+. o.+ |
|      =o oo+= |
|      . ++.+E+ |
|      . . =o=oo |
|              . |
|      .oo = o o |
|      oo.O . o |
|      *= + . |
|      oo+o . |
+-----[SHA256]-----+
```

NEVER EVER SHARE THIS ONE.

### ssh-rsa

```
AAAAB3NzaC1yc2EAAAADAQABAAQCAQC1QUpBAwXo+13PSbMBQHUSc3jYfZFTn3K8mggecfBDir5SoQ
g6vNmji4/7bvQoQOgJs2wT7SKuigG2WAQUZDKuf/drXts8Mc1yZJIYBIUXVM6N24IT3V+HpD4kX/BJOPP
xs4TLb+++Fgs6Jv+nS4VwWxxhvA9fgpKHZEctZqGX/zdNU3H5pWF/yrrGkuPp0SfIE4rnHbg1ga0Ci6O2g0sr
UVUCXzC7qvMa0IHc8NpE+GtPMkbjawAOotIP6V0UZf6ycswo8GyeLwJ602ARej9HsMmg1bp2EYKksMS
CGmdi5wa8Yip9ZlnP5XpFQ4POupmdPBupV2bXTEzKMGBT4oY/XKPPURCYJtRNVJ4Y5r7zhXTsofYBrzdG7
cT8kugkG22l8wkkaCFX+ tq3piqMLLJ8K2JDwdUK7UISmtNA6/lr1xH+kZFa4Wq2acmDs11UiSofbPboDbQt
6PMWTuhkMF877Cg5hp+5HQwoFMjQVAsGeiYwDml3AOGv1TTRjF6qt9C489tVh+MG3JB6R4CoFwkz+O
CphdZP62JPrjx+8i0PEBFPfL7CiFE9hFAI9CQJPCjRz/loXiBrrawAcxPrehDsYGIw== myusername@my_host
```



## Adding your SSH key to your GitLab profile.

<https://git.biohpc.swmed.edu/help/ssh/index.md>

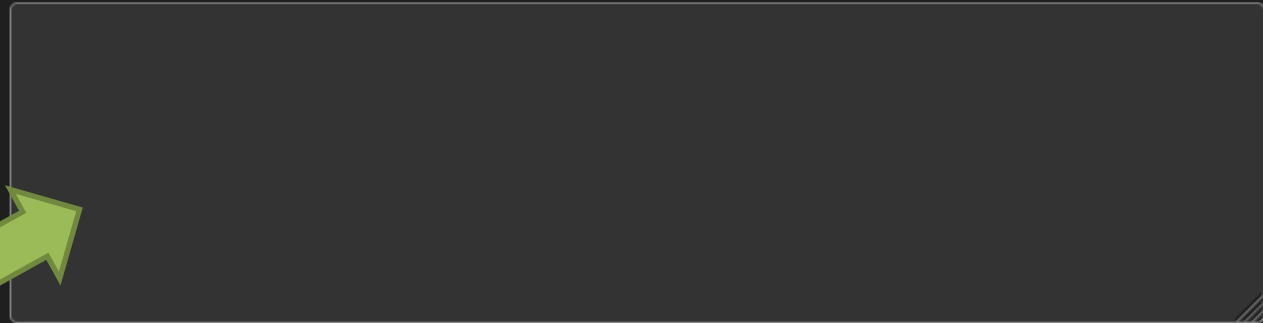
### SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

#### Add an SSH key

Add an SSH key for secure access to GitLab. [Learn more.](#)

#### Key



Begins with 'ssh-rsa', 'ssh-dss', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'.

#### Title

e.g. My MacBook key

Give your individual key a title. This will be publicly visible.

#### Expiration date

mm / dd / yyyy

Key becomes invalid on this date.

Add key

#### ssh-rsa

```
AAAAB3NzaC1yc2EAAAADAQABAAQCAQC1QUpBAwXo+13PSbMBQHUSc3jYfZFTn3K8mggec
fBDir5SoQg6vNmji4/7bvQoQOgJs2wT7SKuigG2WAQUZDKuf/drXts8Mc1yZJJYBIUXVM6N24
IT3V+HpD4kX/BJ0PPxs4TLb+jKb89SNXOBYN20D1QRDprQ+BEzFASdp2/xByUfjWtKcZcqcXlnh
IjTD4kzQI8uayh2n5aB6JcKhVLOTNv++Fgs6Jv+nS4VwWxxhvA9fgpKHZEctZqGX/zdNU3H5pWF
/yrGkuPp0SflE4rnHbg1ga0Ci6O2g0srUV/XKPPURCYJtRNVJ4Y5r7zhXTsofYBrzdG7cT8kugkG2
2I8wkkaCFX+tg3piqMLLJ8K2JDwdUK7UISmtNA6/lr1xH+kZFa4Wq2acmDs11UiSofbPboDbQt
6PMWTuhkMF877Cg5hp+5HQwoFMjQVAsGeiYwDml3AOGv1TTRJF6qt9C489tVh+MG3JB6R4
CoFwkz+OCphdZP62JPrx+8i0PEBFPfL7CiFE9hFAI9CQJPCjRz/IoXiBrrawAcxPrehDsYGIw==
myusername@my_host
```

## Creating a new remote repository

```
$ git remote add new_origin git@git.biohpc.swmed.edu:<user_name>/my_new_project.git
$ git remote -v
new_origin      git@git.biohpc.swmed.edu:<user_name>/my_new_project.git (fetch)
new_origin      git@git.biohpc.swmed.edu:<user_name>/my_new_project.git (push)
$ git push new_origin master
Counting objects: 23, done.
Delta compression using up to 32 threads.
Compressing objects: 100% (13/13), done.
Writing objects: 100% (23/23), 2.17 KiB | 0 bytes/s, done.
Total 23 (delta 2), reused 0 (delta 0)
remote:
remote:
remote: The private project <user_name>/my_new_project was successfully created.
remote:
remote: To configure the remote, run:
remote:   git remote add origin git@git.biohpc.swmed.edu:<user_name>/my_new_project.git
remote:
remote: To view the project, visit:
remote:   https://git.biohpc.swmed.edu/<user_name>/my_new_project
remote:
remote:
remote:
To git@git.biohpc.swmed.edu:<user_name>/my_new_project.git
* [new branch]      master -> master
```

## Further Reading

---

- Various hands-on resources: <https://docs.github.com/en/get-started/quickstart/git-and-github-learning-resources>
- Further resources, from Atlassian: <https://www.atlassian.com/git>
- A little on why Git can be useful for describing your projects:
  - <https://jeremykun.com/2020/01/14/the-communicative-value-of-using-git-well/>
- A more in-depth examination of the abstract ideas behind Git: <https://think-like-a-git.net/>
- The ‘behind-the-scenes’ of how Git works at its most basic level: <https://jwiegley.github.io/git-from-the-bottom-up/>
- If you listen to podcasts, Coding Blocks has a subseries where they discuss this e-book at length and from several different perspectives.
  - <https://www.codingblocks.net/podcast/git-from-the-bottom-up-the-index/>

---

Thank you!

[web] [portal.biohpc.swmed.edu](http://portal.biohpc.swmed.edu)

[email] [biohpc-help@utsouthwestern.edu](mailto:biohpc-help@utsouthwestern.edu)