Version Control with Git

Feng Chen
HPC User Services
LSU HPC & LONI
sys-help@loni.org

Louisiana State University
Baton Rouge
February 16, 2022

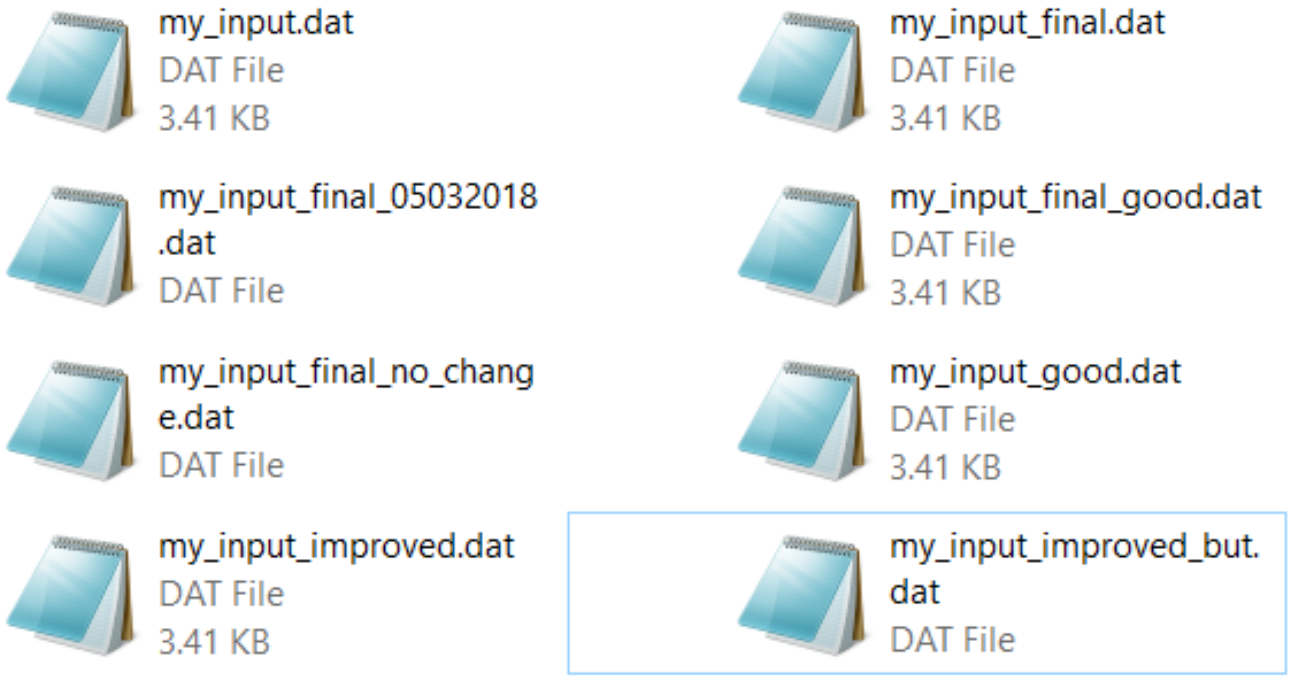
Outline

- **Why we need Git?**
  - Some background
- **Run Git locally**
  - Create a Git repository
  - Working directory and staging area
  - Manage changes
- **Working with Git branch**
  - Create and merge branches
  - Conflict
  - Branch management strategies
- **Working with remote repository**
  - Github

Why do we need a Version Control System?

One Simple Reason…

Copy and Rename approach…??



What is the problem in the previous slide?

- **Copy files into another directory / make a tarball / rename with timestamps**
  - `src_implicit`
  - `src_explicit_works`
  - `src_explicit_fails`
  - `src_20180503.tgz`
- **This approach is very common because it is so *simple*.**

- ***However:***
  - It's incredibly error prone
  - Too easy to accidentally overwrite or delete files.
  - Complex to manage.
  - Difficult to compare differences in history.
  - Hard to collaborate with others.
- **Start to learn good habits now.**

Levels of Version Control

chaoseed Follow

Level 5: Private Github repository with access only for you and your team
Level 4: Public Github repository where random people send you strange pull requests
Level 3: Automated backup program saves your files to an external server on a regular basis
Level 2: Back up your files to a flash drive every now and then
Level 1: Copy the source file and add "_old_v2" to the name
Level 0: Copy the line and comment it out

From: https://nixcraft.tumblr.com/post/177998927172/levels-of-version-control

What is a Version Control System (VCS)

- **Definition**
  - Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

What can a VCS do for you?

- **A VCS can:**
  - Records changes to a file or set of files over time so that you can recall specific versions later.
  - Revert specific files or the entire project back to a previous state.
  - Compare changes over time.
  - See who made the changes.
- **Not just for source code:**
  - LaTeX files
  - Text files
  - Configuration files
  - Input files for a code

- ***Caution:***
  - most VCSs don't handle binary files well.
    - Git LFS
    - Dropbox?

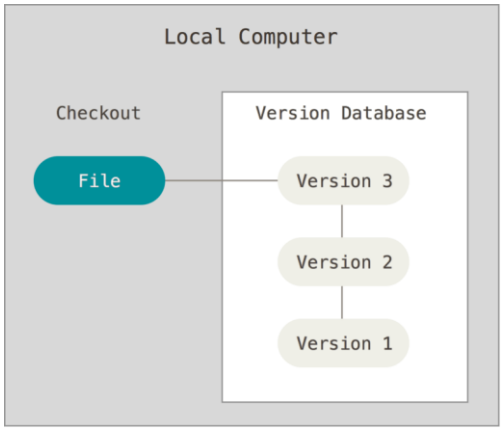
Advantages of using VCS

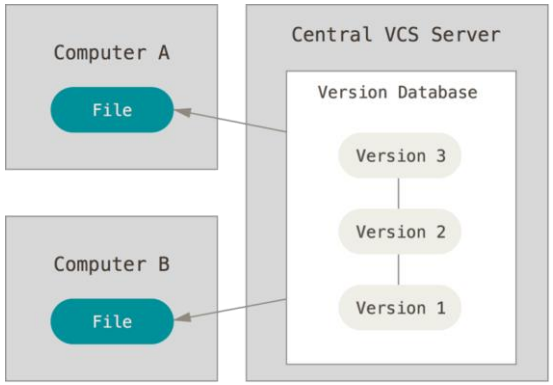
- **Using VCS is liberating**
  - It makes it easy to recover from mistakes.
  - It remember what you did for you.

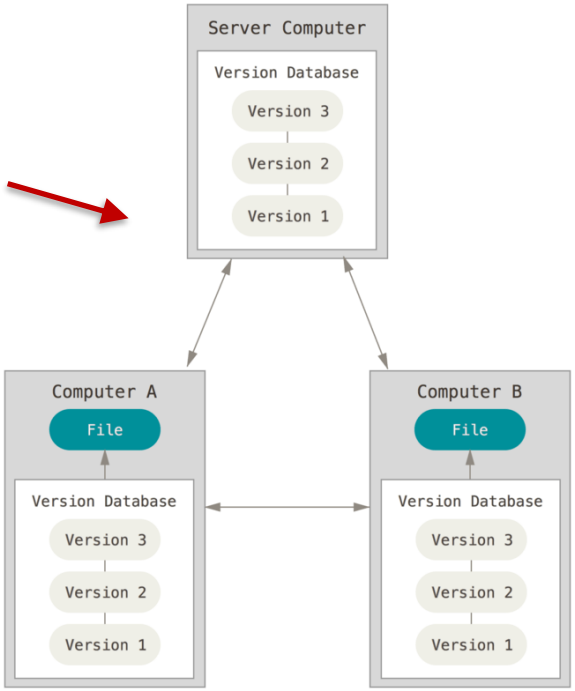
- ***Best of all*:**
  - It's easy to use.
  - Not easy to lose information.

Types of VCS

- **Different types of VCSs**
  - Local VCS
    - GNU RCS
  - Centralized VCS
    - CVS
    - SVN
  - Distributed VCS
    - **Git**
    - Mercurial
    - hg







Advantages of DVCS

- **You *don't need* an internet connection to interact with the repo.**
- **Duplication: Every clone is really a full backup of all the data. If any server dies any of the client repositories can be copied back up to the server to restore it.**
- **Supports multiple remote repositories, so you can collaborate with different groups of people.**
  - many of the DVCS systems deal pretty well with having several remote repositories they can work with,
  - This allows you to set up several types of workflows that aren't possible in centralized systems, such as hierarchical models.

History of Git

- **Created in 2005 by Linus Torvalds, the creator of Linux, with the goals of:**
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)

```
The name "git" was given by Linus Torvalds when he wrote the very
first version. He described the tool as "the stupid content tracker"
and the name as (depending on your way):

 - random three-letter combination that is pronounceable, and not actually used by any common
UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
 - stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
 - "global information tracker": you're in a good mood, and it actually works for you. Angels
sing, and a light suddenly fills the room.
 - "goddamn idiotic truckload of sh*t": when it breaks
```

Ref: https://en.wikipedia.org/wiki/Git

Run Git locally

Log onto SuperMIC and load git module

```
ssh <username>@smic.hpc.lsu.edu
```

```
[fchen14@smic1 ~]$ module av git

----------------------------
/usr/local/packages/Modules/default/modulefiles/linux-rhel7-ivybridge -----
------------------------
git/2.25.0/intel-19.0.5
[fchen14@smic1 ~]$ module load git
Autoloading libidn2/2.1.1a/intel-19.0.5
Autoloading pcre2/10.23/intel-19.0.5
[fchen14@smic1 ~]$ which git
/usr/local/packages/git/2.25.0/sbiqd4kw/bin/git
```

First time Git setup

- **Set up your identity. Especially important when you work with other people:**

```
[fchen14@smic1 ~]$ git config --global user.name "Feng Chen"
[fchen14@smic1 ~]$ git config --global user.email fchen14@lsu.edu
```

- **Checking your settings:**

```
[fchen14@smic1 ~]$ git config --list
user.name=Feng Chen
user.email=fchen14@lsu.edu
core.editor=vi
core.excludesfile=exclude_git.pattern
push.default=matching
```

Getting help from Git

- **Different syntax for getting help from command line:**

```
git help <verb>
git <verb> --help
man git-<verb>
```

- **For example:**

```
[fchen14@smic1 ~]$ git help config
[fchen14@smic1 ~]$ git config --help
[fchen14@smic1 ~]$ man git-config
```

- **Example output:**

```
Git-CONFIG(1)                    Git Manual                    Git-CONFIG(1)
NAME
       git-config - Get and set repository or global options
SYNOPSIS
       git config [<file-option>] [type] [-z|--null] name [value [value_regex]]
       git config [<file-option>] [type] --add name value
       git config [<file-option>] [type] --replace-all name value [value_regex]
```

- **More common help source: *google it!*** 

Git Basic Usage

- **What is a repository?**
  - A directory (`.git/`) contains all information regarding the history of your code.
- **Creating a Git repository from an existing directory**

```
$ git init
```

```
[fchen14@smic1 ~]$ mkdir myrepo
[fchen14@smic1 ~]$ cd myrepo
[fchen14@smic1 myrepo]$ git init
Initialized empty Git repository in /home/fchen14/myrepo/.git/
[fchen14@smic1 myrepo]$ ls .git
branches  config  description  HEAD  hooks  info  objects  refs
```

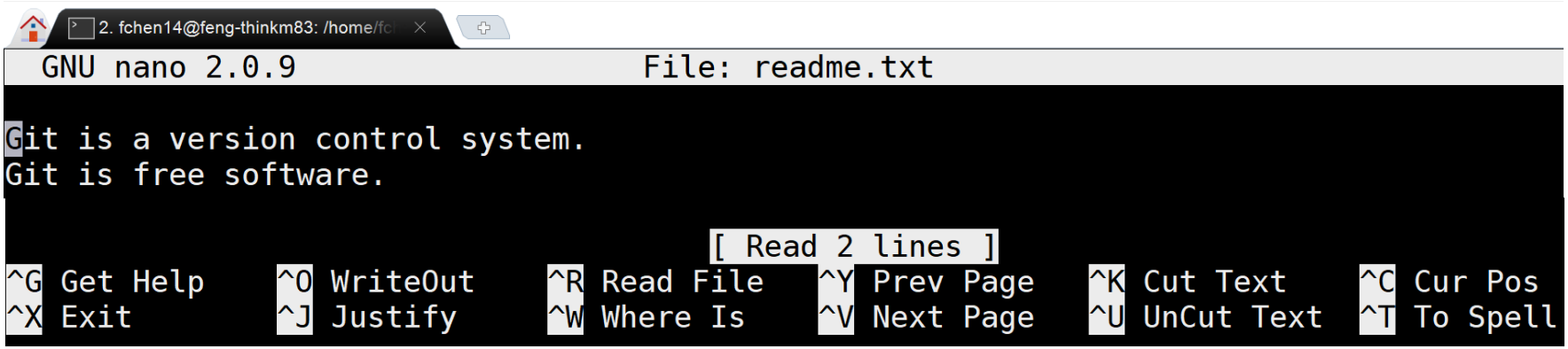
Prepare a readme.txt for the repo

- **Create a readme file in the directory ~/myrepo**

```
[fchen14@smic1 myrepo]$ nano readme.txt
```

- **Add the below two lines of text (using your favorite editor: vi, emacs or nano) to "readme.txt":**

```
Git is a version control system.
Git is free software.
```



Add “readme.txt” to the repository

- **First add the file to the repository**

```
[fchen14@smic1 myrepo]$ git add readme.txt
```

- **Commit the file to the repository**

```
[fchen14@smic1 myrepo]$ git commit -m "added a readme file"
[master (root-commit) 666c968] added a readme file
 1 file changed, 2 insertions(+)
 create mode 100644 readme.txt
```

- **Some explanations**
  - `git add`: add the readme.txt to the staging area (index)
  - `git commit -m`: commit your changes to the repo with a message (“-m”)

View New Changes

- **Now change "readme.txt" to the below contents:**

```
Git is a distributed version control system.
Git is free software.
```

- **Use `git status` to check our results:**

```
[fchen14@smic1 myrepo]$ 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:   readme.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

- **If you do not remember the changes, use `git diff`**

```
[fchen14@smic1 myrepo]$ git diff
diff --git a/readme.txt b/readme.txt
index 46d49bf..9247db6 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,2 +1,2 @@
-Git is a version control system.
+Git is a distributed version control system.
 Git is free software.
```

Commit new changes to the repository

- **Same as before, two steps, git add and then git commit:**

```
[fchen14@smic1 myrepo]$ git add readme.txt
[fchen14@smic1 myrepo]$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        modified:   readme.txt
[fchen14@smic1 myrepo]$ git commit -m "added word distributed"
[master 08cb0ef] added word distributed
 1 file changed, 1 insertion(+), 1 deletion(-)
[fchen14@smic1 myrepo]$ git status
On branch master
nothing to commit, working tree clean
```

Commit another change to the repository

- **Now change "readme.txt" to the below contents:**

```
Git is a distributed version control system.
Git is free software distributed under the GPL.
```

- **Add and commit your changes:**

```
[fchen14@smic1 myrepo]$ nano readme.txt
...add the above changes...
[fchen14@smic1 myrepo]$ git add readme.txt
[fchen14@smic1 myrepo]$ git commit -m "appended GPL"
[master 4dd4cf0] appended GPL
 1 file changed, 1 insertion(+), 1 deletion(-)
```

- **So far, we have 3 versions added to the Git repository "myrepo"**

Review the version history

- **Use `git log` to review our version history**

```
[fchen14@smic1 myrepo]$ git log
commit 4dd4cf05696809c3bd26e3a0ed5c5f2e9aea765a (HEAD -> master)
Author: Feng Chen <fchen14@lsu.edu>
Date:   Sat May 5 10:31:38 2018 -0500
    appended GPL
commit 08cb0ef326f5d149cae16ad9201022ea056eafe5
Author: Feng Chen <fchen14@lsu.edu>
Date:   Sat May 5 10:06:32 2018 -0500
    added word distributed
commit 666c96898be1ae630c4d958d55482bba9d4516d5
Author: Feng Chen <fchen14@lsu.edu>
Date:   Sat May 5 09:12:40 2018 -0500
    added a readme file
```

- **Or you can use `git log --oneline` for a short version**

```
[fchen14@smic1 myrepo]$ git log --oneline
4dd4cf0 (HEAD -> master) appended GPL
08cb0ef added word distributed
666c968 added a readme file
```

Some concepts/terminology

- **Commit**
  - Records changes to the repository identified by a SHA-1 hash
- **HEAD**
  - A special pointer in Git called HEAD. In Git, this is a pointer to the local branch you're currently on.
- **SHA-1 hash**
  - Git uses a checksum mechanism called SHA-1 hash to differentiate and name the commits.
  - This is a 40-character string composed of hexadecimal characters (0–9 and a–f) and calculated based on the contents of a file or directory structure in Git.
  - Something you would get with:

```
[fchen14@smic1 myrepo]$ shasum readme.txt
2b1eb3f6006e80d38bbb176fab0747f224c48c03  readme.txt
```

  - You will see these hash values all over the place in Git. In fact, Git stores everything in its database not by file name but by the hash value of its contents.

Add a useful command before “Travel”

- **First, add a useful git command alias by copy & pasting the following command in your terminal:**

```
$ git config --global alias.graph 'log --all --oneline --decorate --graph'
```

- **Then type “`git graph`” in your terminal:**

```
[fchen14@smic1 myrepo]$ git graph
* 4dd4cf0 (HEAD -> master) appended GPL
* 08cb0ef added word distributed
* 666c968 added a readme file
```

- **We will explain the detailed meanings of the command later.**

Jump to previous versions

- **Go back to the version "added word distributed"**

```
[fchen14@smic1 myrepo]$ git graph
* 4dd4cf0 (HEAD -> master) appended GPL
* 08cb0ef added word distributed
* 666c968 added a readme file
[fchen14@smic1 myrepo]$ git reset --hard HEAD^
HEAD is now at 08cb0ef added word distributed
```

- **Or you can directly use the SHA-1 hash:**

```
[fchen14@smic1 myrepo]$ git reset --hard 08cb0ef
HEAD is now at 08cb0ef added word distributed
```

- **Then you can take a look at the content of readme.txt:**

```
[fchen14@smic1 myrepo]$ cat readme.txt
Git is a distributed version control system.
Git is free software.
```

What if the previous operation is a mistake?

- **How do I go back to the latest verion?**

```
[fchen14@smic1 myrepo]$ git graph
* 08cb0ef (HEAD -> master) added word distributed
* 666c968 added a readme file
```

- **Hint: we need to find the SHA-1 hash of that commit**
- **Solution: use the command `git reflog`:**

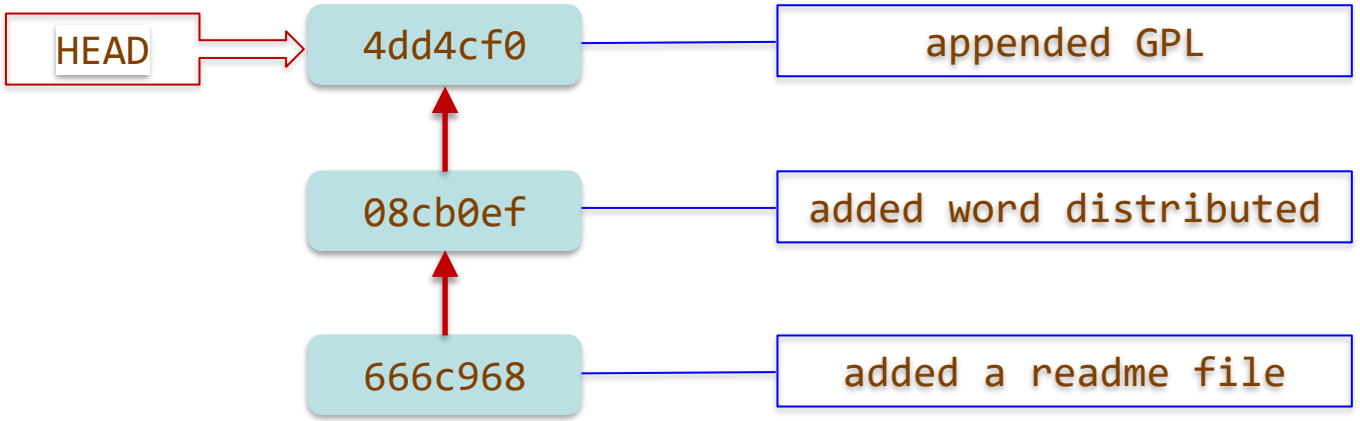
```
[fchen14@smic1 myrepo]$ git reflog
08cb0ef (HEAD -> master) HEAD@{0}: reset: moving to 08cb0ef
08cb0ef (HEAD -> master) HEAD@{1}: reset: moving to HEAD^
4dd4cf0 HEAD@{2}: commit: appended GPL
08cb0ef (HEAD -> master) HEAD@{3}: commit: added word distributed
666c968 HEAD@{4}: commit (initial): added a readme file
```

- **Now we can reset to the specific commit (SHA-1 hash):**

```
[fchen14@smic1 myrepo]$ git reset --hard 4dd4cf0
HEAD is now at 4dd4cf0 appended GPL
[fchen14@smic1 myrepo]$ cat readme.txt #verify the content
Git is a distributed version control system.
Git is free software distributed under the GPL.
```

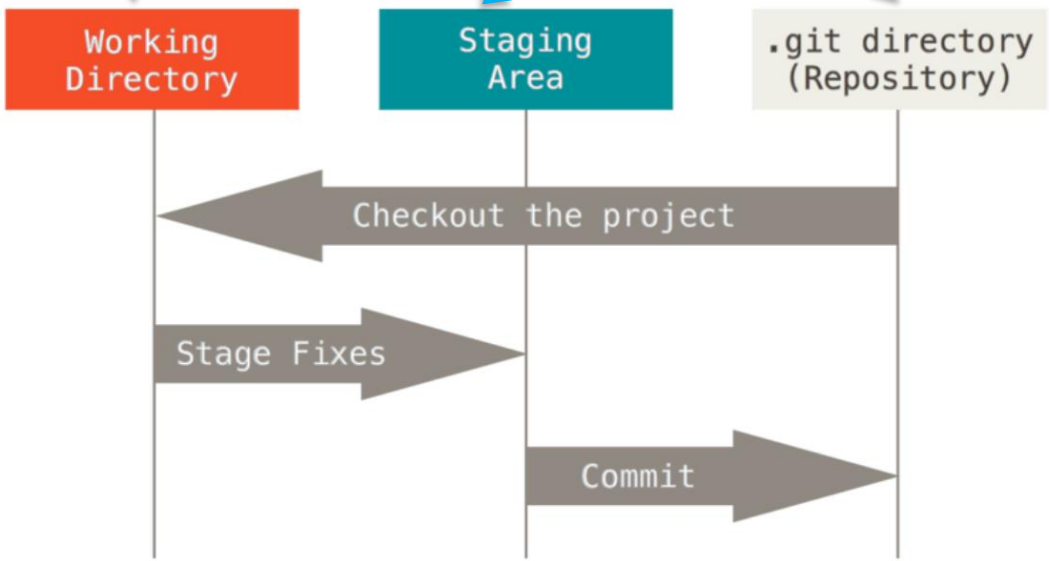
Short Summary - Basic Usage

- **How do we**
  - Initialize a Git repo?
  - Add files to the repo? (two steps)
  - “Travel” between different commits?



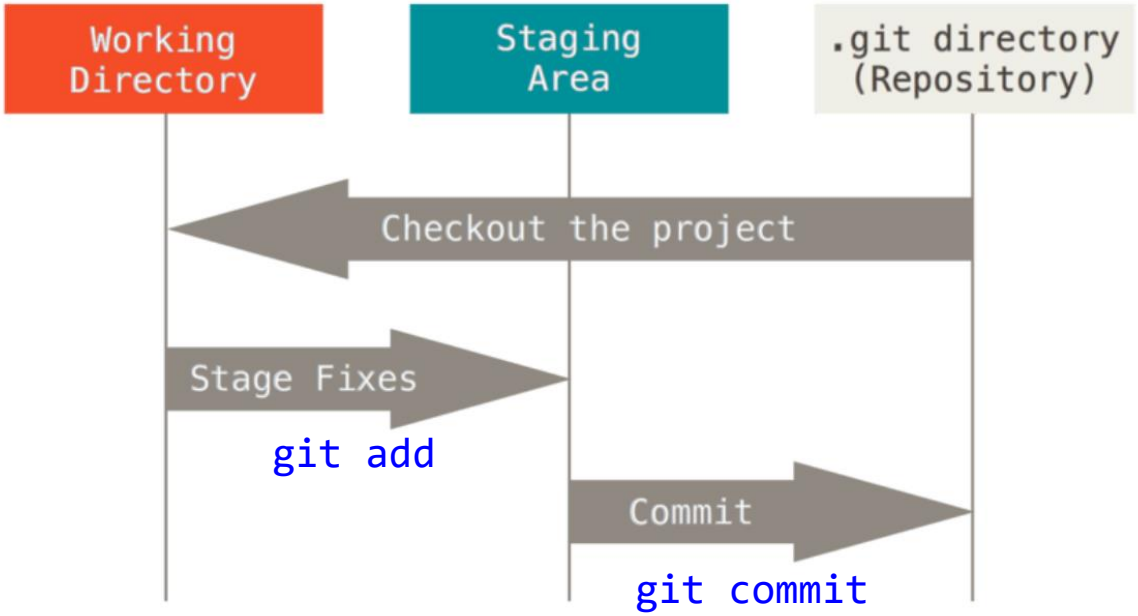
Three main sections of Git

- **The Git directory (`.git/ directory`) where Git stores the meta data and object database for your project.**
- **The working directory (working tree) is a single checkout (snapshot) of one version of the project, i.e. the working directory consist of files that you are currently working on (you see).**
- **The staging area is a file that stores information about what will go into your next commit. It's sometimes referred to as the "index", but it's also common to refer to it as the staging area.**



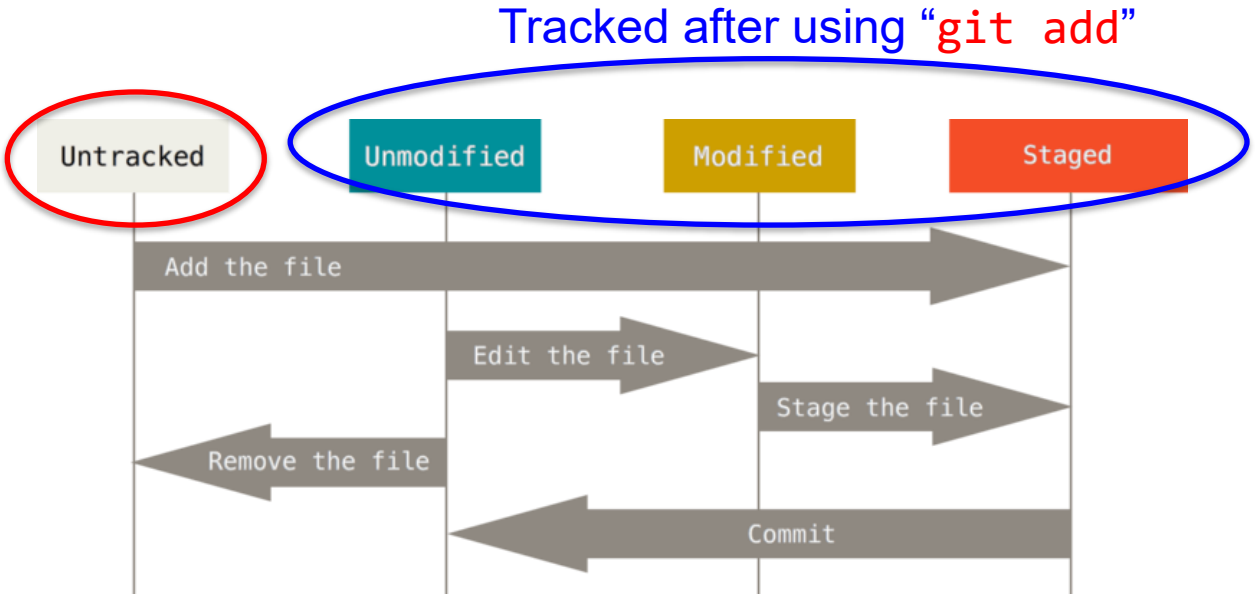
Basic Git workflow

- **You modify files in your working directory.**
  1. `git add`: You stage the files, adding snapshots of them to your staging area.
  2. `git commit`: You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.



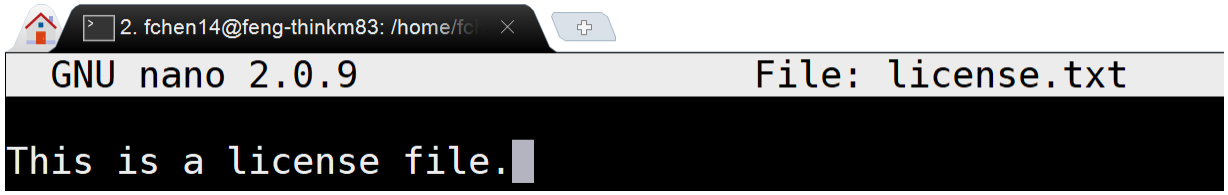
Recording Changes to the Repository

- **Git records *changes* to the repository.**
- **Two types of files:**
  - Untracked
  - Tracked
- **Two types of changes**
  - Adding a new, previously untracked file
  - Modification of a file already under Git tracking



Understanding the Staging area

- **Let's do two things to our repo:**
  1. Add a new file (license.txt, content can be arbitrary) to the repository:



  2. Add the below line to the readme.txt:

     Git has a mutable index called stage.

- **Then use the git status to check our repo status:**

```
[fchen14@smic1 myrepo]$ 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:   readme.txt
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        license.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

Only add new file to staging area

- **Now we only add license.txt to the staging area:**

```
[fchen14@smic1 myrepo]$ git add license.txt
[fchen14@smic1 myrepo]$ git commit -m "add license"
[master c183956] add license
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 license.txt
```

- **Then check the working directory status**

```
[fchen14@smic1 myrepo]$ 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:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

- The change of readme.txt is not committed, why?

Managing the changes - Example (1)

- **Make the following changes to readme.txt**
  1. Add a newline to readme.txt: `Git tracks changes.`

     ```
     [fchen14@smic1 myrepo]$ nano readme.txt
     ```
  2. Add readme.txt to staging area

     ```
     [fchen14@smic1 myrepo]$ git add readme.txt
     ```
  3. Change the last line of readme.txt to: `Git tracks changes of files.`

     ```
     [fchen14@smic1 myrepo]$ nano readme.txt
     ```
  4. Commit the changes

     ```
     [fchen14@smic1 myrepo]$ git commit -m "Git tracks changes"
     [master 662a255] Git tracks changes
      1 file changed, 1 insertion(+)
     ```
  5. Check the status of the repo

     ```
     [fchen14@smic1 myrepo]$ 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:   readme.txt
     no changes added to commit (use "git add" and/or "git commit -a")
     ```

Managing the changes - Example (2)

- **What does `git commit` do?**
  - Recall the last workflow:
    - 1st change -> add -> 2nd change -> commit
  - Answer: only changes added to the staging area will be committed!
- **Use `git diff` to view the differences between the version in the working directory and the repository:**

```
[fchen14@smic1 myrepo]$ git diff -- readme.txt
diff --git a/readme.txt b/readme.txt
index 76d770f..a9c5755 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,4 +1,4 @@
 Git is a distributed version control system.
 Git is free software distributed under the GPL.
 Git has a mutable index called stage.
-Git tracks changes.
+Git tracks changes of files.
```

❖ What should be the correct steps for adding the 2nd change?

Three usages of "`git add`"

- **Trace new file**
  - Put file under "git radar"
- **Add changes of files to staging area**
- **Resolve conflict**
  - Explain later...

Short Summary - Staging Area

- **What are the differences between:**
  - Working directory
  - Staging area
  - Repository

Undoing Things

- **How to cancel a change? e.g. what if you need to delete the last line of readme.txt?**

```
[fchen14@mike2 myrepo]$ cat readme.txt
Git is a distributed version control system.
Git is free software distributed under the GPL.
Git has a mutable index called stage.
Git tracks changes of files.
Don't know why my boss still prefers SVN.
```

Cancel the changes

- **Three situations:**
  - Discard the changes in the working directory.

```
$ git checkout -- <filename>
[fchen14@smic1 myrepo]$ git checkout -- readme.txt
[fchen14@smic1 myrepo]$ cat readme.txt
Git is a distributed version control system.
Git is free software distributed under the GPL.
Git has a mutable index called stage.
Git tracks changes of files.
```

  - Discard the changes added to the staging area. (2 steps)

```
$ git reset HEAD <filename>
$ git checkout -- <filename>
[fchen14@smic1 myrepo]$ git reset HEAD readme.txt
Unstaged changes after reset:
M       readme.txt
[fchen14@smic1 myrepo]$ git checkout -- readme.txt
```

  - Discard the changes that is already committed?

```
$ git reset --hard <commit-hash>
```

Delete file(s)

- **Let's first add a file to the repository:**

```
[fchen14@smic1 myrepo]$ touch test.txt
[fchen14@smic1 myrepo]$ git add test.txt
[fchen14@smic1 myrepo]$ git commit -m "add test.txt"
```

  - I do want to remove this file from the repository

```
[fchen14@smic1 myrepo]$ rm test.txt
[fchen14@smic1 myrepo]$ 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:    test.txt
no changes added to commit (use "git add" and/or "git commit -a")
[fchen14@smic1 myrepo]$ git rm test.txt # what is the alternative
command?
rm 'test.txt'
[fchen14@smic1 myrepo]$ git commit -m "test.txt"
[master cdef552] test.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 test.txt
```

  - I deleted the file by mistake? (`git checkout -- test.txt`)

Move (Rename) files

- **Recover the deleted test.txt and rename it to testnew.txt**

```
[fchen14@smic1 myrepo]$ git checkout HEAD^1 test.txt
[fchen14@smic1 myrepo]$ ls
license.txt  readme.txt  test.txt
[fchen14@smic1 myrepo]$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        new file:   test.txt
[fchen14@smic1 myrepo]$ git mv test.txt testnew.txt
[fchen14@smic1 myrepo]$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        new file:   testnew.txt
[fchen14@smic1 myrepo]$ git commit -m "renamed test.txt to testnew.txt"
[master 83963bd] renamed test.txt to testnew.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 testnew.txt
```

Short Summary - Change management

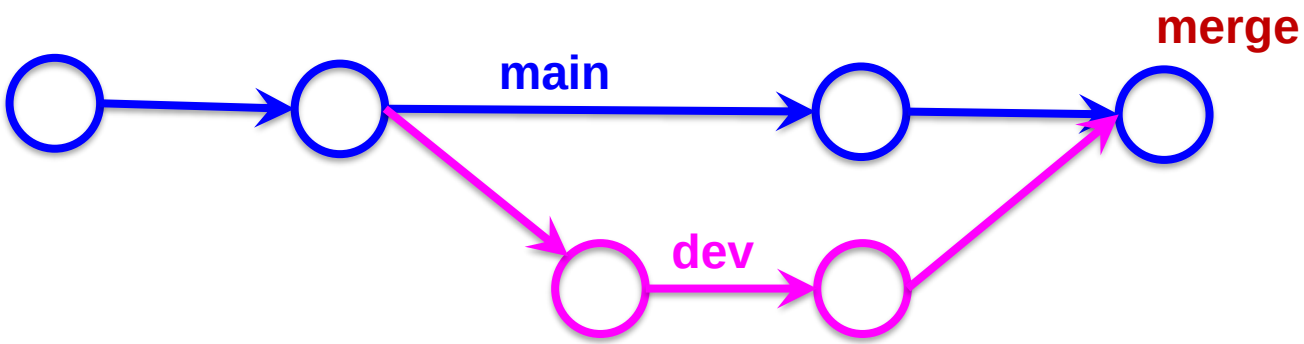
- **Understanding the staging area**
- **Discard changes in 3 different cases**
  - In working directory
  - In staging area
  - Committed
- **Delete and move files**

Git locally

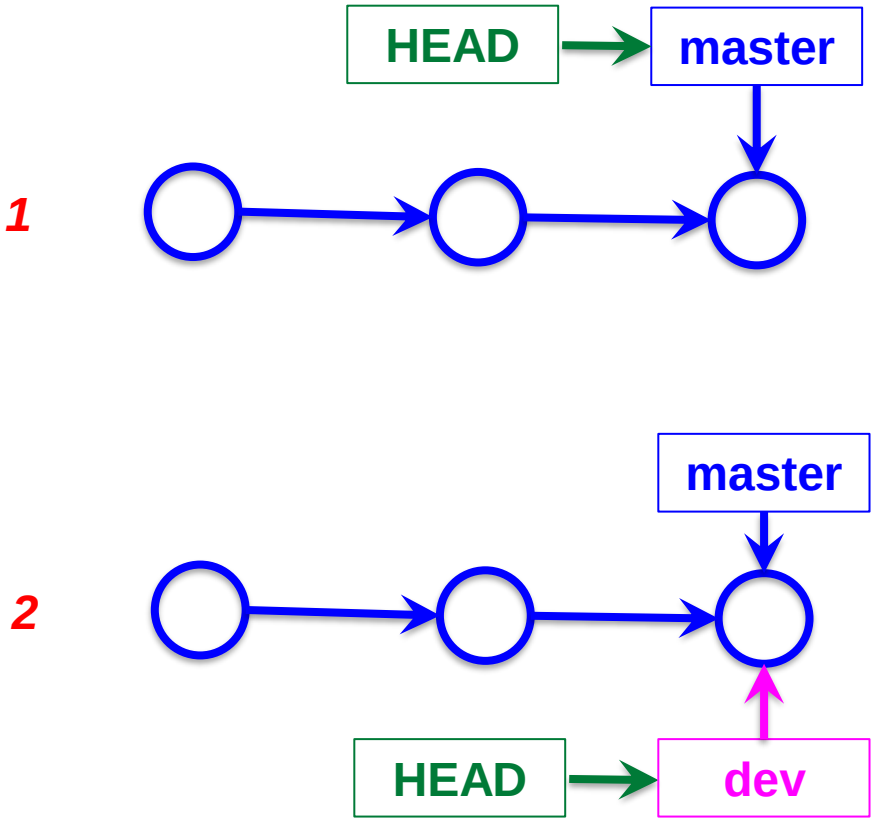
Working with Git branch

Introduction to Git branch

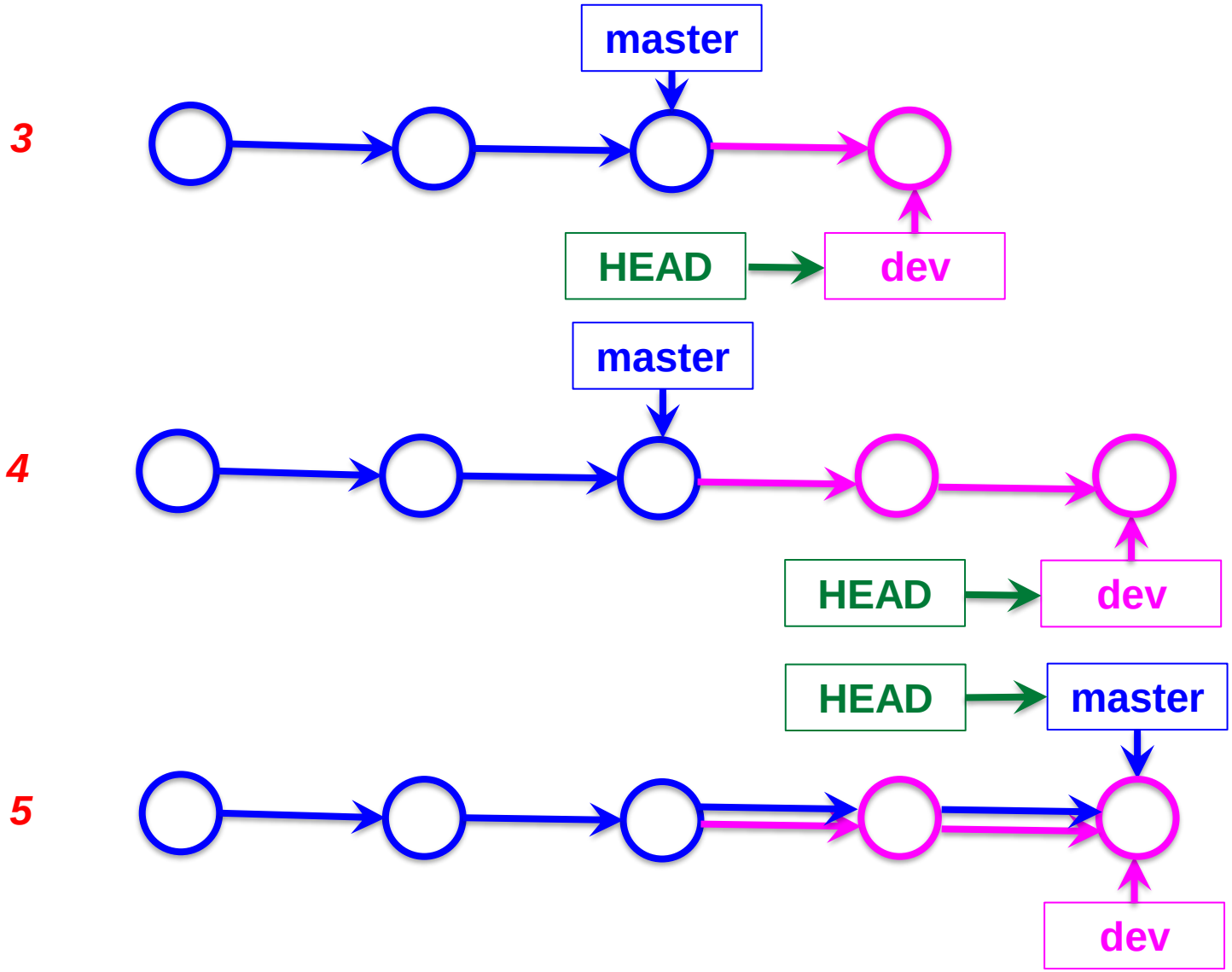
- **A new scenario:**
  - Need to develop a new feature, need 2 weeks
  - The new feature will interfere with the current functions
  - Need to let the new feature separate from the main branch
- *"Some people refer to Git's branching model as its "killer feature," and it certainly sets Git apart in the VCS community. Why is it so special? The way Git branches is incredibly lightweight, making branching operations nearly instantaneous, and switching back and forth between branches generally just as fast. Unlike many other VCSs, Git encourages workflows that branch and merge often, even multiple times in a day. Understanding and mastering this feature gives you a powerful and unique tool and can entirely change the way that you develop."*
  - Ref: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell



Create and Merge branches (1)



Fast-Forward Merge



Create a dev branch and commit

- **Create a dev branch and switch to that branch**

```
[fchen14@smic1 myrepo]$ git branch
* master
[fchen14@smic1 myrepo]$ git checkout -b dev
Switched to a new branch 'dev'
[fchen14@smic1 myrepo]$ git branch
* dev
  master
```

- **On dev branch, modify readme.txt by adding a line**

```
Creating a new branch is quick
```

- **Commit the changes:**

```
[fchen14@smic1 myrepo]$ nano readme.txt
[fchen14@smic1 myrepo]$ git add readme.txt
[fchen14@smic1 myrepo]$ git commit -m "branch test"
[dev 6fa8c5f] branch test
 1 file changed, 1 insertion(+)
```

Check the difference between branches

- **Switch to the master branch and check the readme.txt**

```
[fchen14@smic1 myrepo]$ git checkout master
Switched to branch 'master'
[fchen14@smic1 myrepo]$ cat readme.txt
Git is a distributed version control system.
Git is free software distributed under the GPL.
Git has a mutable index called stage.
Git tracks changes of files.
```

- **This verifies the change happens only on the dev branch**

Merge dev branch to master branch

- **Merge the work on the dev branch to the master branch:**

```
[fchen14@smic1 myrepo]$ git branch
  dev
* master
[fchen14@smic1 myrepo]$ git graph # a pre-defined alias
* 6fa8c5f (dev) branch test
* 83963bd (HEAD -> master) renamed test.txt to testnew.txt
* e9ee24a removed test.txt
...
[fchen14@smic1 myrepo]$ git merge dev
Updating 83963bd..6fa8c5f
Fast-forward
 readme.txt | 1 +
 1 file changed, 1 insertion(+)
```

- **Verify the branch status using our pre-defined command alias**

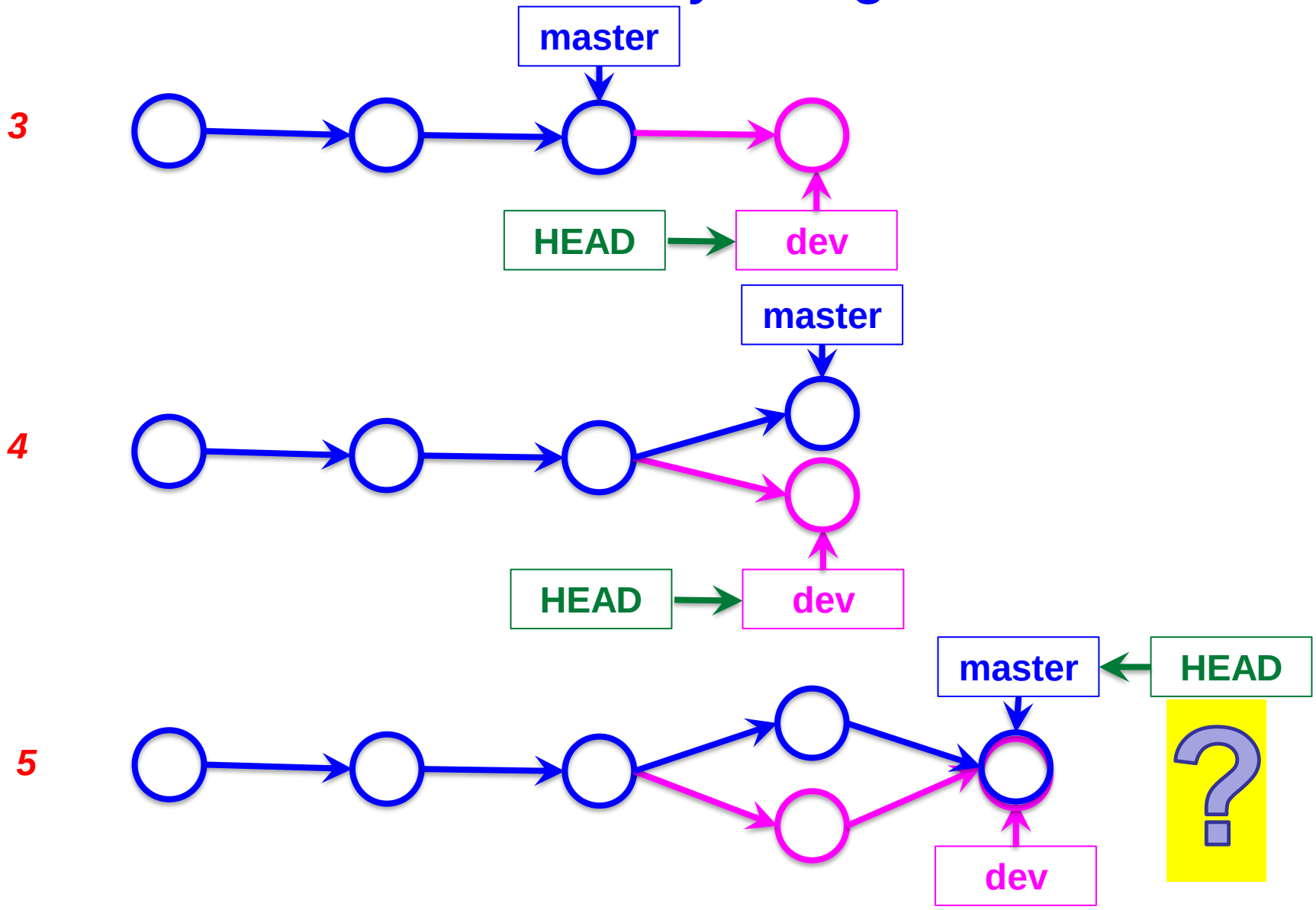
```
[fchen14@smic1 myrepo]$ git graph
* 6fa8c5f (HEAD -> master, dev) branch test
* 83963bd renamed test.txt to testnew.txt
* e9ee24a removed test.txt
* 575744a add test.txt
...
```

Delete branch after merge

- **It's safe to delete the branch after merge**

```
[fchen14@smic1 myrepo]$ git branch -d dev
Deleted branch dev (was 6fa8c5f).
[fchen14@smic1 myrepo]$ git branch
* master
[fchen14@smic1 myrepo]$ git graph
* 6fa8c5f (HEAD -> master) branch test
* 83963bd renamed test.txt to testnew.txt
* e9ee24a removed test.txt
...
```

Three-Way Merge



Three way merge and conflict

- **In dev branch, change the last line in readme.txt to:**

  Creating a new branch is quick AND simple.

- **In master branch, change the last line in readme.txt to:**

  Creating a new branch is quick & simple.

```
[fchen14@smic1 myrepo]$ git checkout -b dev
Switched to a new branch 'dev'
[fchen14@smic1 myrepo]$ nano readme.txt # change last line to “AND simple”
[fchen14@smic1 myrepo]$ git add readme.txt
[fchen14@smic1 myrepo]$ git commit -m "AND simple"
[dev 0414374] AND simple
 1 file changed, 1 insertion(+), 1 deletion(-)
[fchen14@smic1 myrepo]$ git checkout master
Switched to branch 'master'
[fchen14@smic1 myrepo]$ nano readme.txt # change last line to “& simple”
[fchen14@smic1 myrepo]$ git add readme.txt
[fchen14@smic1 myrepo]$ git commit -m "& simple"
[master abef850] & simple
 1 file changed, 1 insertion(+), 1 deletion(-)
[fchen14@smic1 myrepo]$ git merge dev
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Resolve conflict manually

- **Use git status to check the conflict files**

```
[fchen14@smic1 myrepo]$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   readme.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

- **Use your favorite editor to check the contents of readme.txt**

```
[fchen14@smic1 myrepo]$ cat readme.txt
Git is a distributed version control system.
Git is free software distributed under the GPL.
Git has a mutable index called stage.
Git tracks changes of files.
<<<<<<< HEAD
Creating a new branch is quick & simple.
=======
Creating a new branch is quick AND simple.
>>>>>>> dev
```

Commit after resolving the conflicts

- **Manually resolve the conflicts indicated by Git**

```
<<<<<<< HEAD
=======
>>>>>>> dev
```

- **We will resolve this conflict by changing the last line to:**

```
Creating a new branch is quick and simple.
```

- **Commit your changes to complete the merge process.**

```
[fchen14@smic1 myrepo]$ git add readme.txt
[fchen14@smic1 myrepo]$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)
Changes to be committed:
        modified:   readme.txt
[fchen14@smic1 myrepo]$ git commit -m "resolve conflict"
[master e732763] resolve conflict
```

Verify git status and delete dev branch

- **Verify git branch graph**

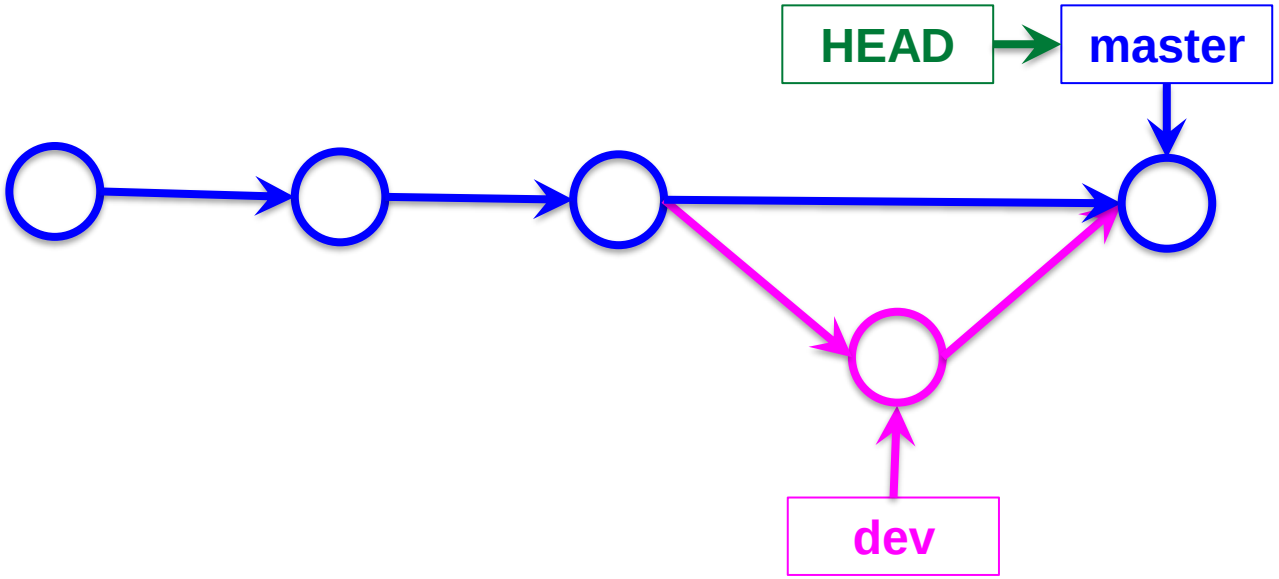
```
[fchen14@smic1 myrepo]$ git graph
*   e732763 (HEAD -> master) resolve conflict
|\
| * 0414374 (dev) AND simple
* | abef850 & simple
|/
* 6fa8c5f branch test
```

- **Delete the dev branch**

```
[fchen14@smic1 myrepo]$ git branch -d dev
Deleted branch dev (was 0414374).
[fchen14@smic1 myrepo]$ git graph
*   e732763 (HEAD -> master) resolve conflict
|\
| * 0414374 AND simple
* | abef850 & simple
|/
* 6fa8c5f branch test
```

Branch Strategy

- **Fast-forward merge will lose branch information when the branch is deleted.**
- **It is suggested to use `--no-ff` even for a fast-forward merge so that the branch information is retained even after branch deletion.**

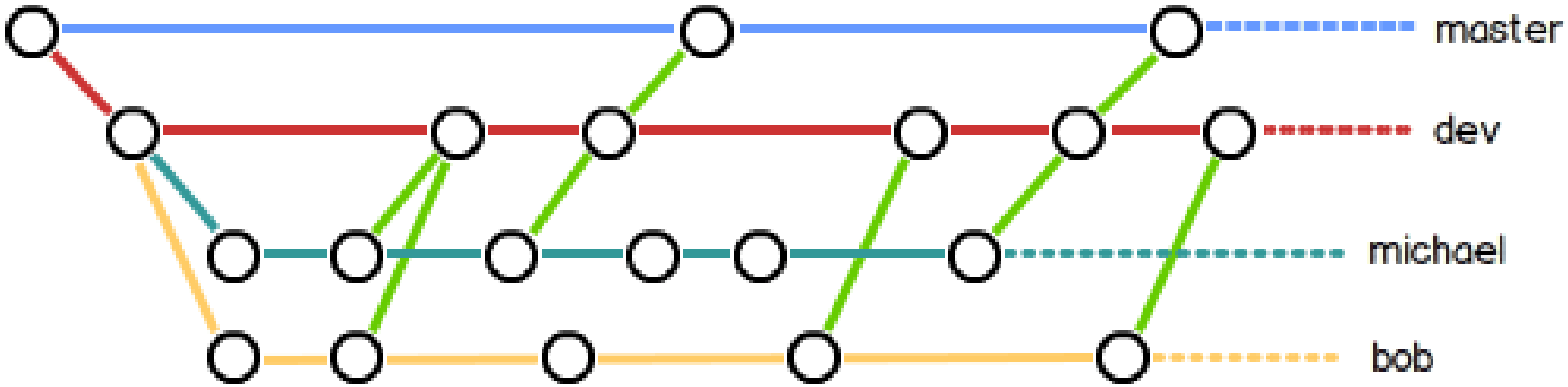


Using --no-ff merge

```
[fchen14@smic1 myrepo]$ git checkout -b dev
Switched to a new branch 'dev'
[fchen14@smic1 myrepo]$ nano readme.txt # add a line
[fchen14@smic1 myrepo]$ git add readme.txt
[fchen14@smic1 myrepo]$ git commit -m "--no-ff merge"
...
[fchen14@smic1 myrepo]$ git checkout master
Switched to branch 'master'
[fchen14@smic1 myrepo]$ git merge --no-ff -m "merge with no-ff" dev
Merge made by the 'recursive' strategy.
 readme.txt | 1 +
 1 file changed, 1 insertion(+)
[fchen14@smic1 myrepo]$ git graph
*   f12ac0f (HEAD -> master) merge with no-ff
|\
| * 03c1409 (dev) --no-ff merge
|/
*   e732763 resolve conflict
```

Group development branch

- **Master branch is always stable**
- **All development work in dev branch, merge to master when necessary**
- **Every developer has his/her own branch.**



Bug branch and git stash

- **Interrupted workflow**
  - When you are in the middle of something, your boss comes in and demands that you fix something immediately.

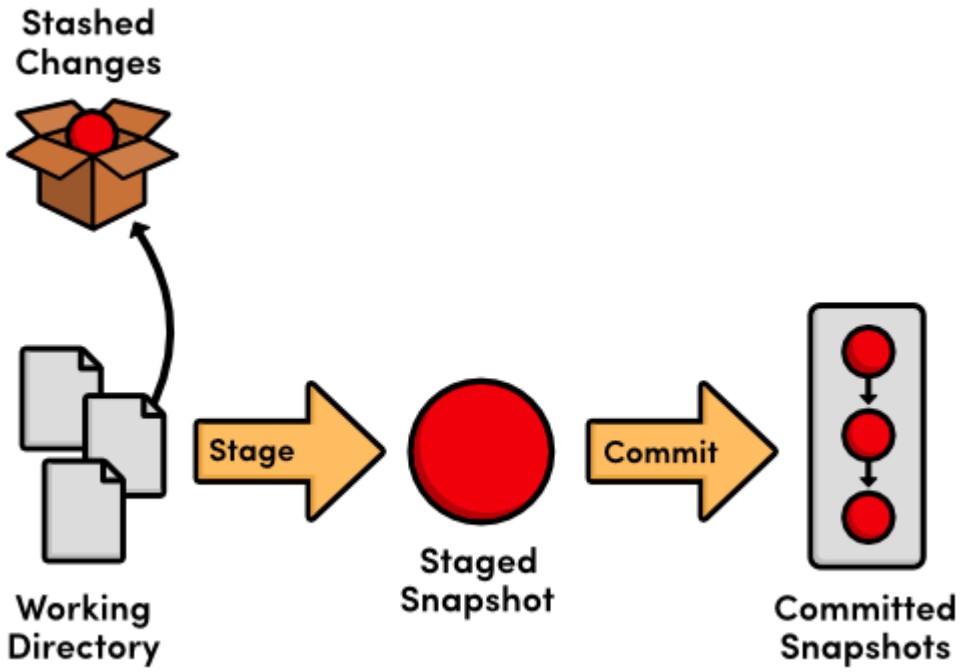
```
# ... hack hack hack ...
$ git checkout -b my_wip
$ git commit -a -m "WIP"
$ git checkout master
$ edit emergency fix
$ git commit -a -m "Fix in a hurry"
$ git checkout my_wip
$ git reset --soft HEAD^
# ... continue hacking ...
```

  - You can use git stash to simplify the above, like this:

```
# ... hack hack hack ...
$ git stash
$ edit emergency fix
$ git commit -a -m "Fix in a hurry"
$ git stash pop
# ... continue hacking ...
```

A visualization of “git stash”

- **Ref:** https://code.tutsplus.com/tutorials/quick-tip-leveraging-the-power-of-git-stash--cms-22988



Use git stash (1)

- **Use git stash when you want to record the current state of the working directory and the index, but want to go back to a clean working directory. The command saves your local modifications away and reverts the working directory to match the HEAD commit.**

```
[fchen14@smic1 myrepo]$ git status
On branch feature
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        modified:   testnew.txt
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        feature.py
[fchen14@smic1 myrepo]$ git add feature.py
[fchen14@smic1 myrepo]$ git stash
Saved working directory and index state WIP on feature: 5b39dd9 bug fix in
readme.txt
[fchen14@smic1 myrepo]$ git checkout master
Switched to branch 'master'
[fchen14@smic1 myrepo]$ echo "fixed bug in main branch." >> readme.txt
[fchen14@smic1 myrepo]$ git add readme.txt
```

Use git stash (2)

```
[fchen14@smic1 myrepo]$ git commit -m "fixed bug in main“
[master a4c6023] fixed bug in main
 1 file changed, 1 insertion(+)
[fchen14@smic1 myrepo]$ git checkout feature
Switched to branch 'feature'
[fchen14@smic1 myrepo]$ git stash list
stash@{0}: WIP on feature: 5b39dd9 bug fix in readme.txt
[fchen14@smic1 myrepo]$ git stash pop
On branch feature
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        new file:   feature.py
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:   testnew.txt

Dropped refs/stash@{0} (04a1c17075630e449d12635d6e22b830e4db3a06)
```

Short Summary
Git branch

- **Two types of merge**
  - Fast-forward
  - Three way
  - Conflict
- **Branch strategy**
- **Interrupted workflow**

Version Control using Git

Working with remote repository

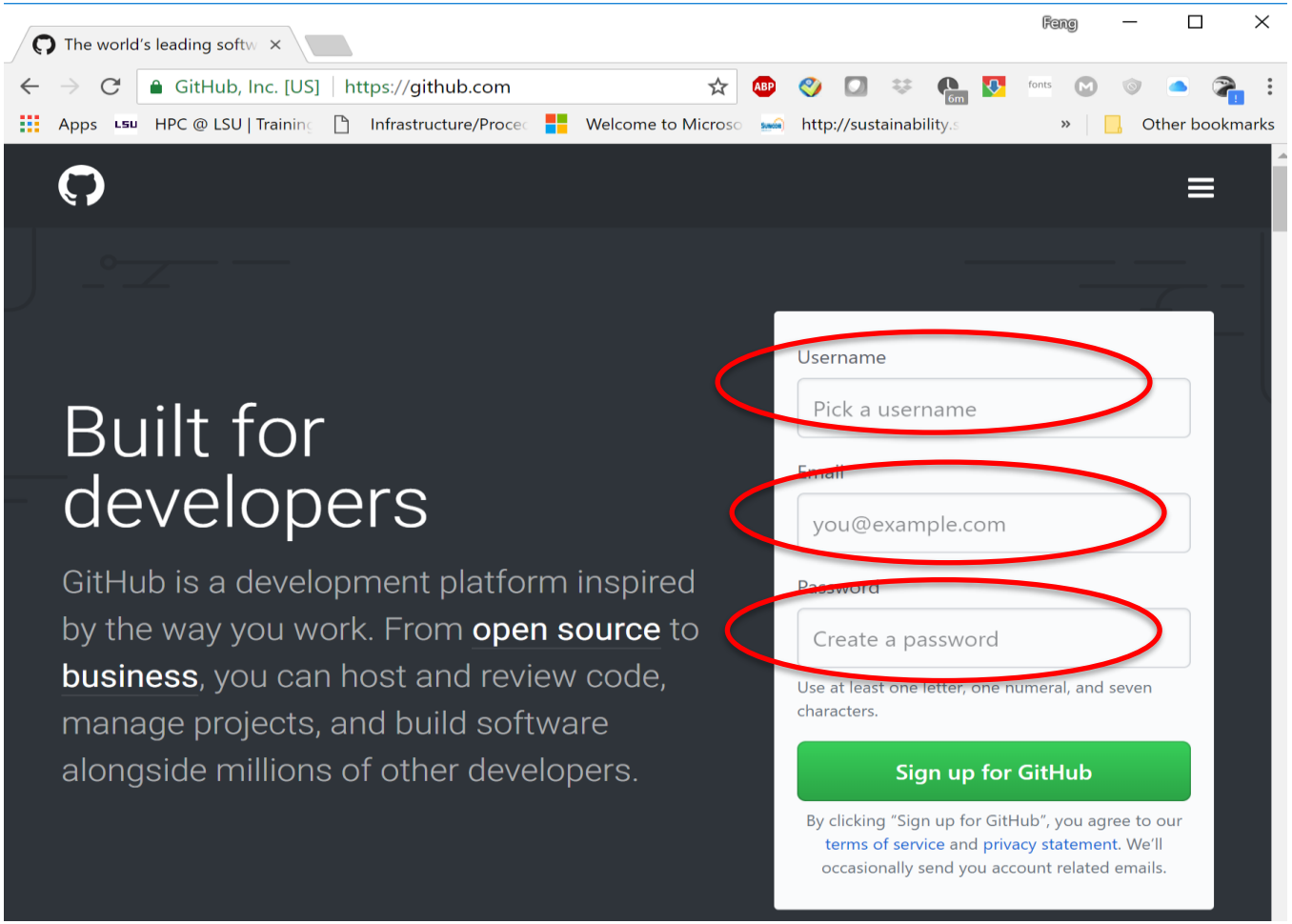
What is a remote repository?

- **Remote repositories are versions of your project that are not on your computer.**
- **They usually include:**
  - the history.
  - branches, called the remote branches.
- **Interaction with remote repositories**
  - You push changes from your computer to the remote.
  - You pull changes from the remote to your computer.

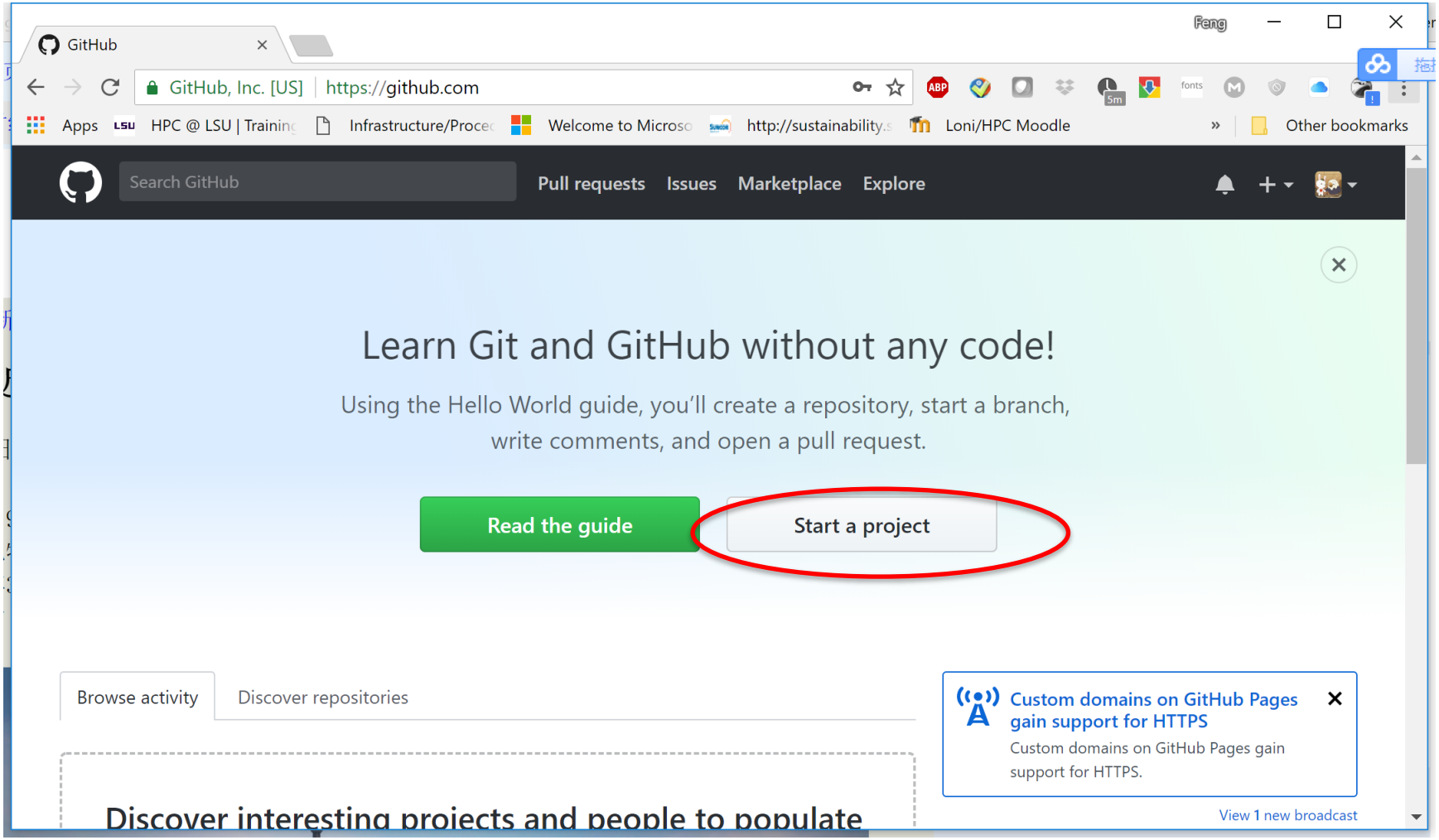
- **Why use remote repositories?**
  - backup your work.
  - Collaborate.
  - You can have several remote repositories.

Signup for GitHub

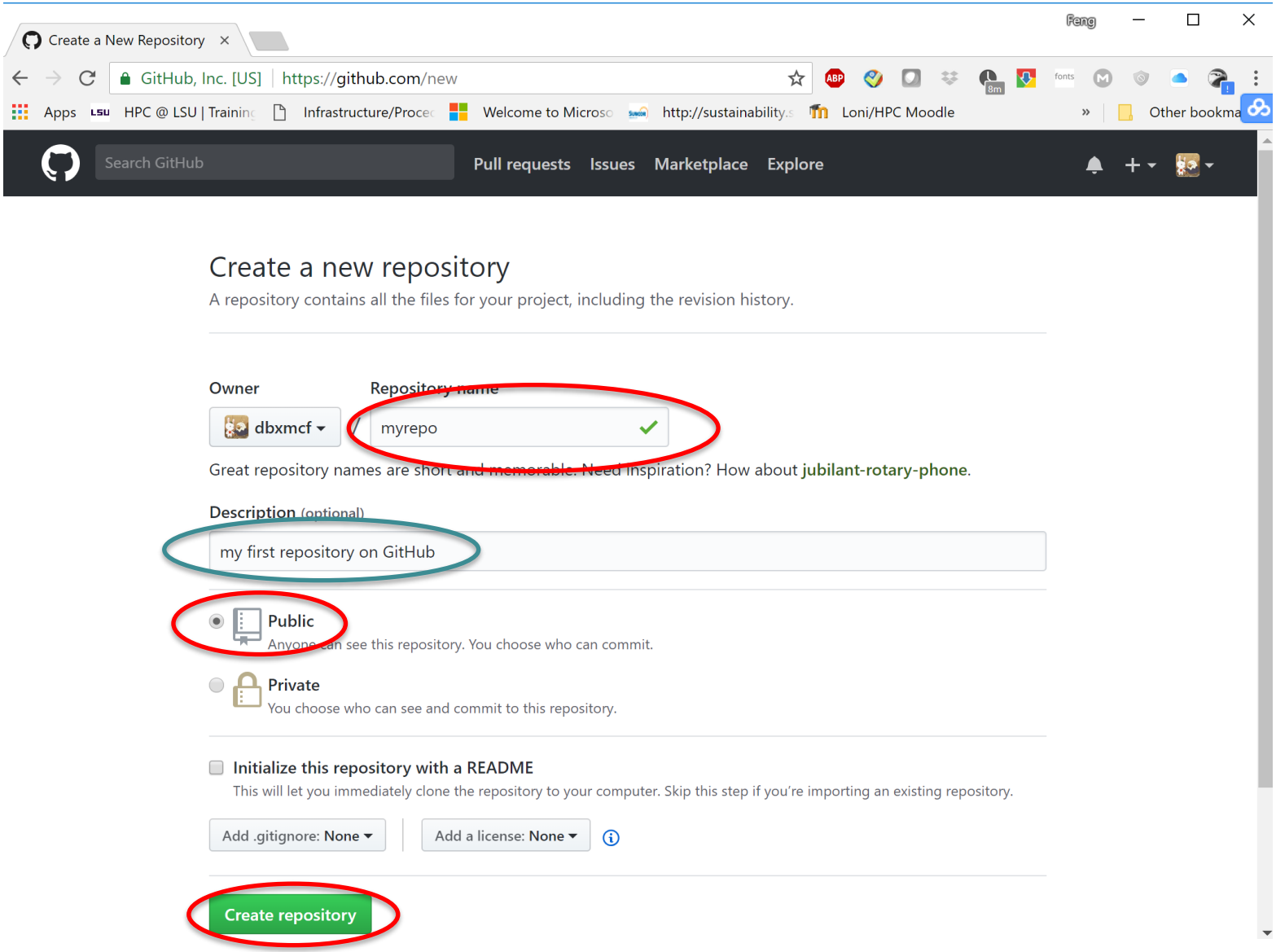
- **Sign up a GitHub account at https://github.com/ if you do not already have one.**



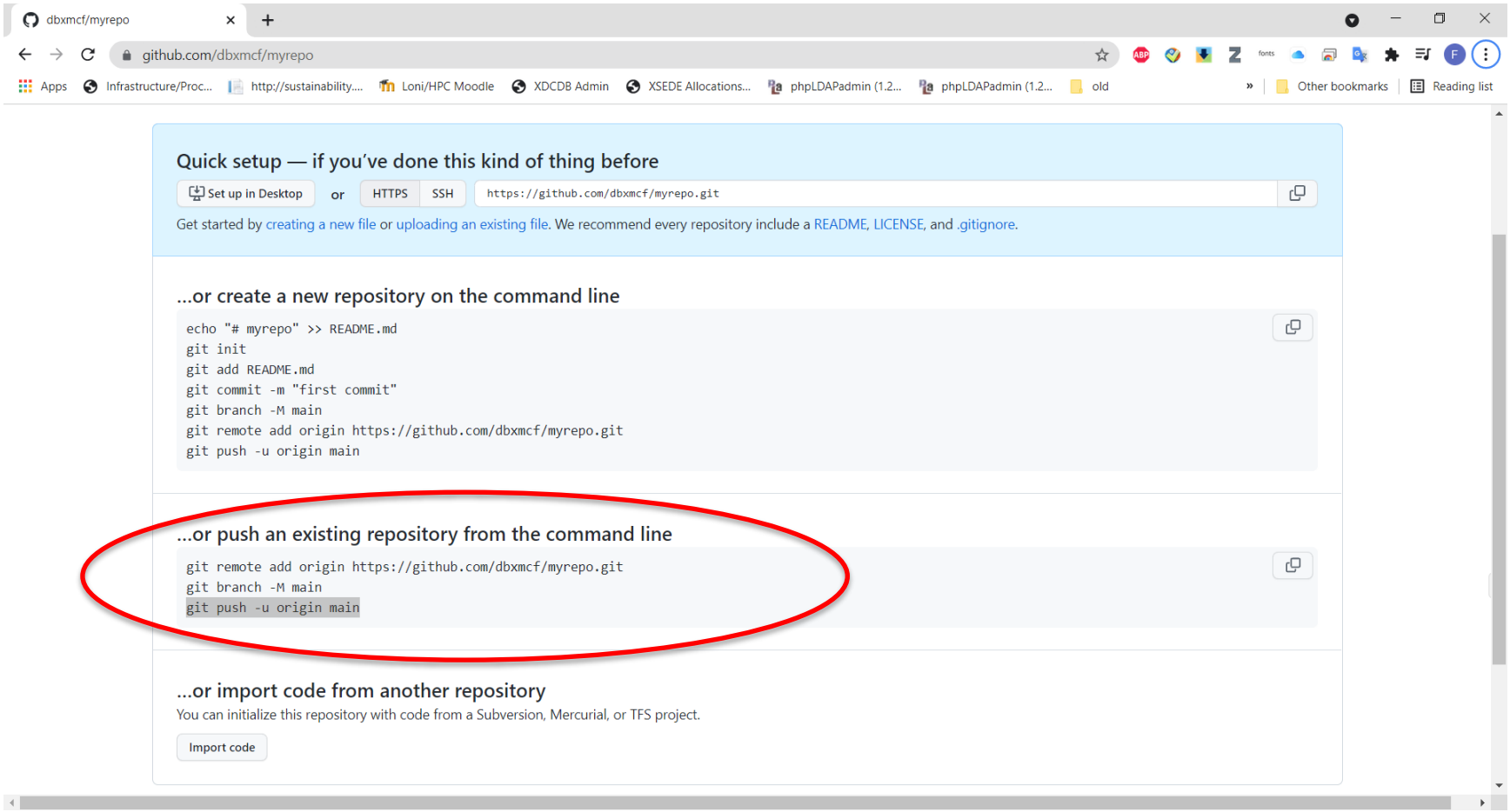
Start a project on GitHub



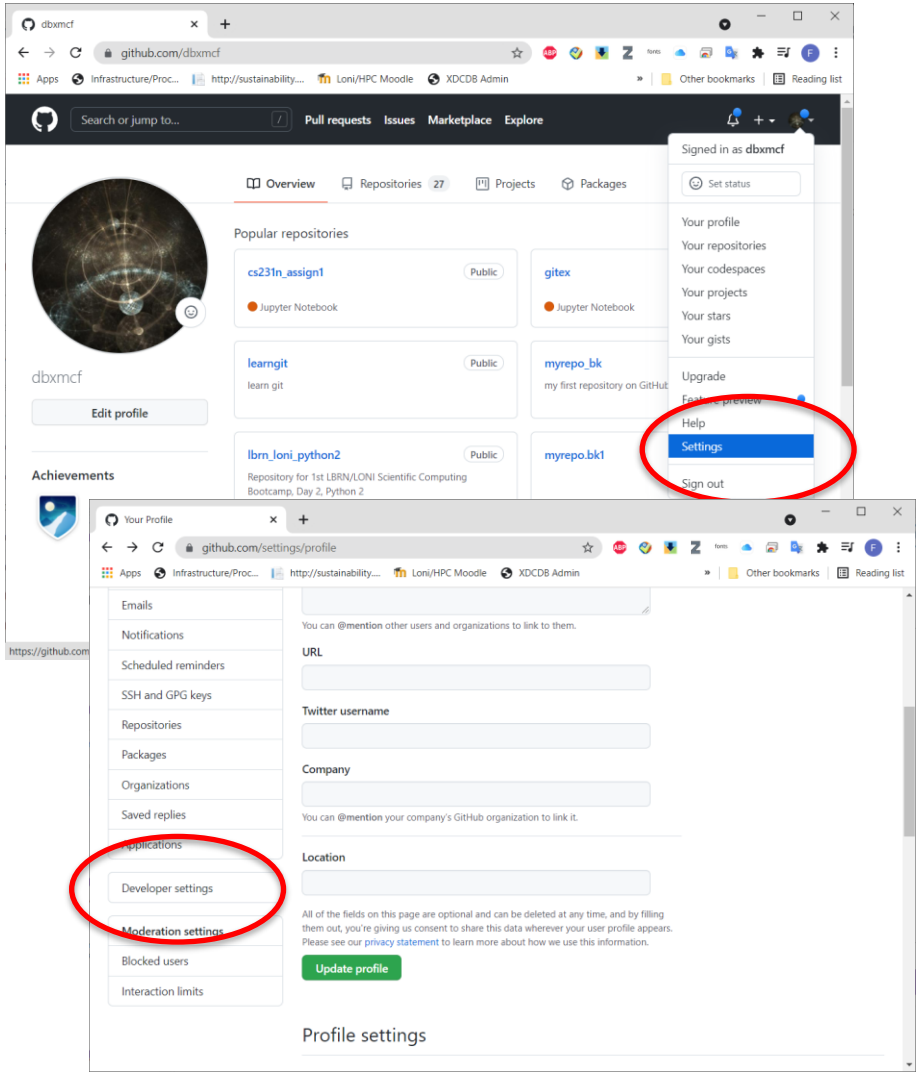
Create a new repository



Push our existing repository

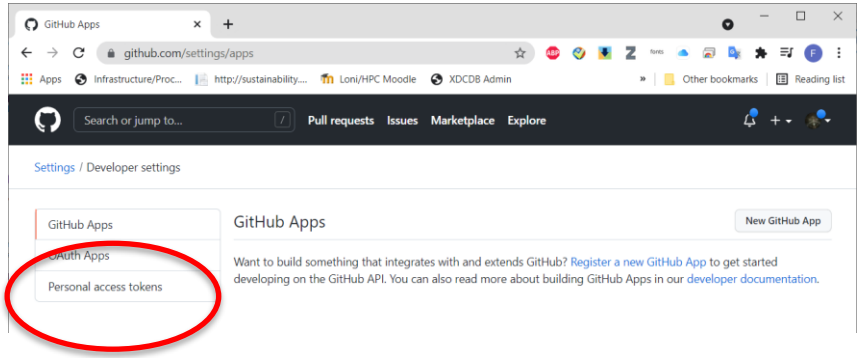


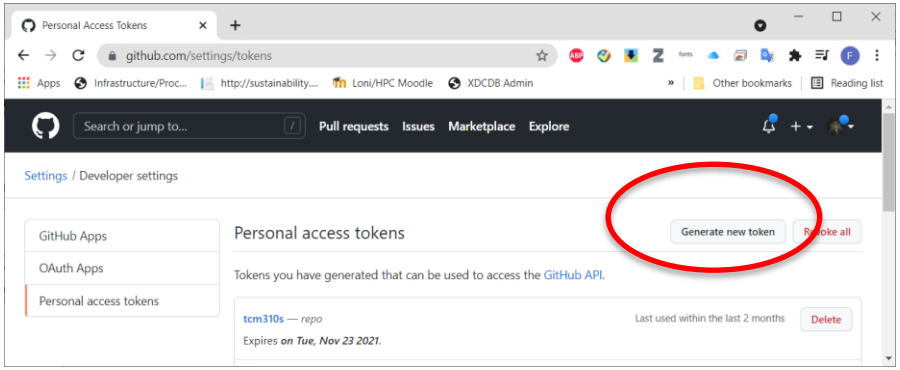
Github now uses personal token



- Click the profile icon and find the settings menu
- In the profile settings page, scroll down to “developer settings”

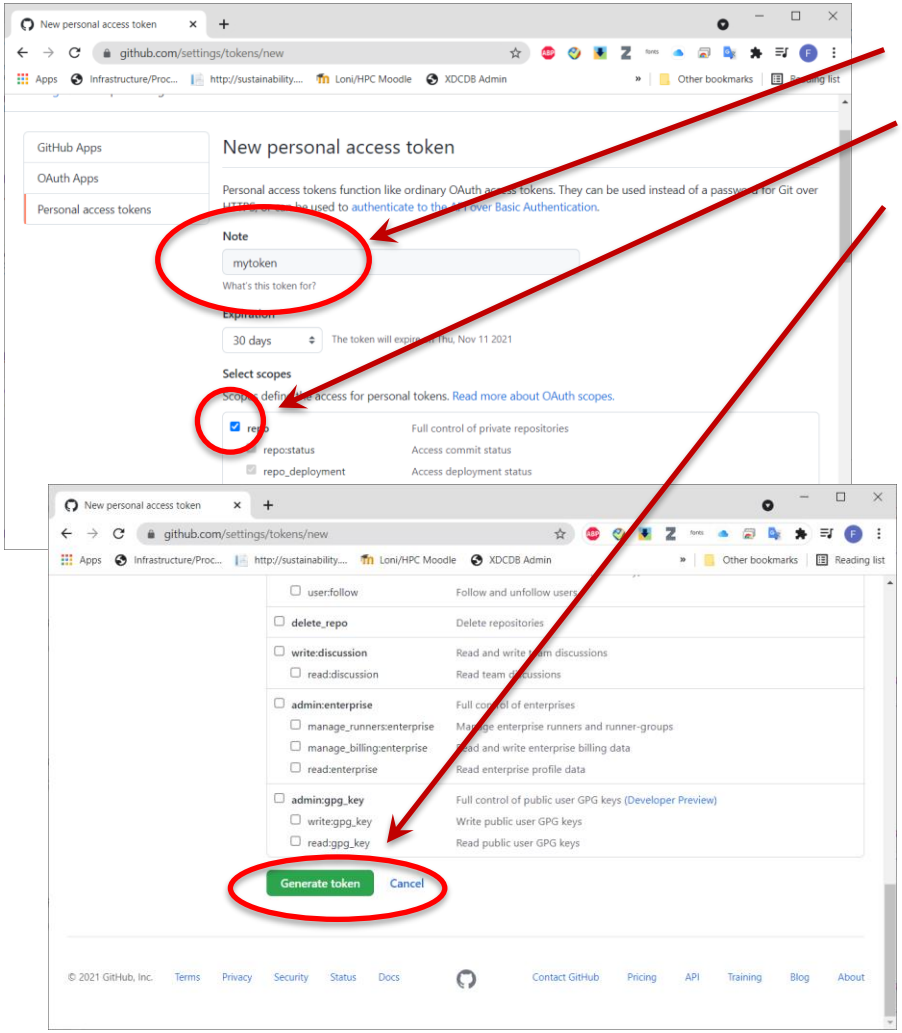
Generate personal token (1)





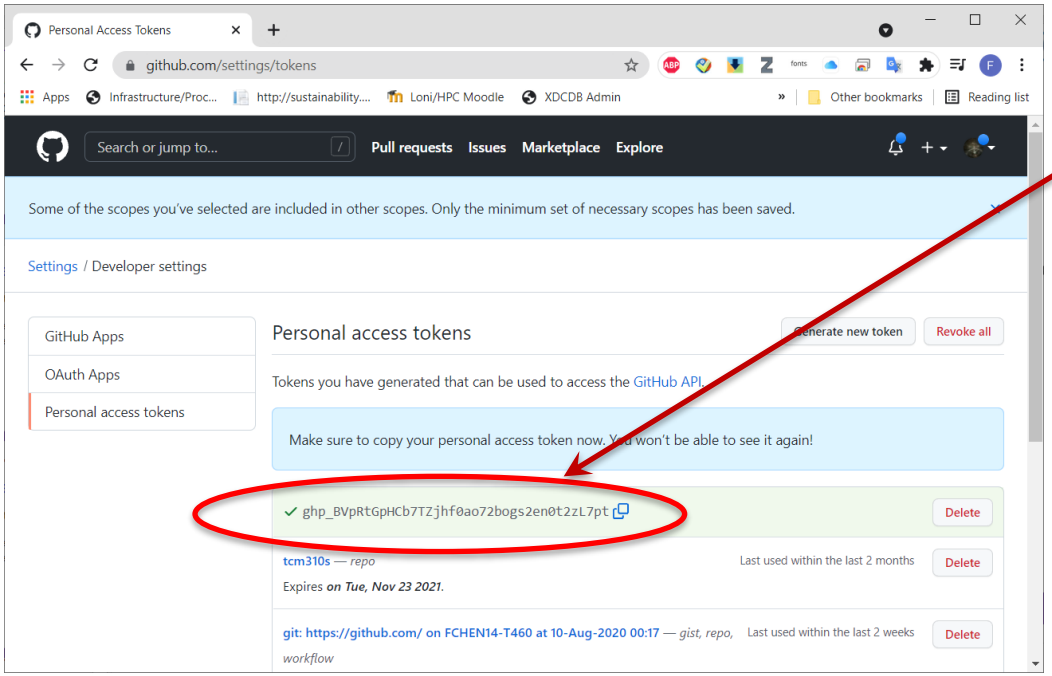
- Click “Personal access tokes”
- And then “Generate new token”

Generate personal token (2)



1. Give a note to the token
2. Check the “repo” box
3. “Generate token”

Generate personal token (3)

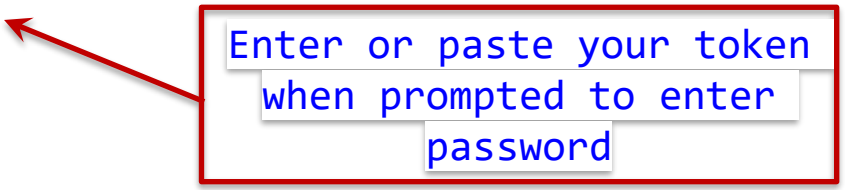


- Copy your token to a safe place, e.g. notepad
- You won't see the token again
- This token will be the password when you push/pull your repository

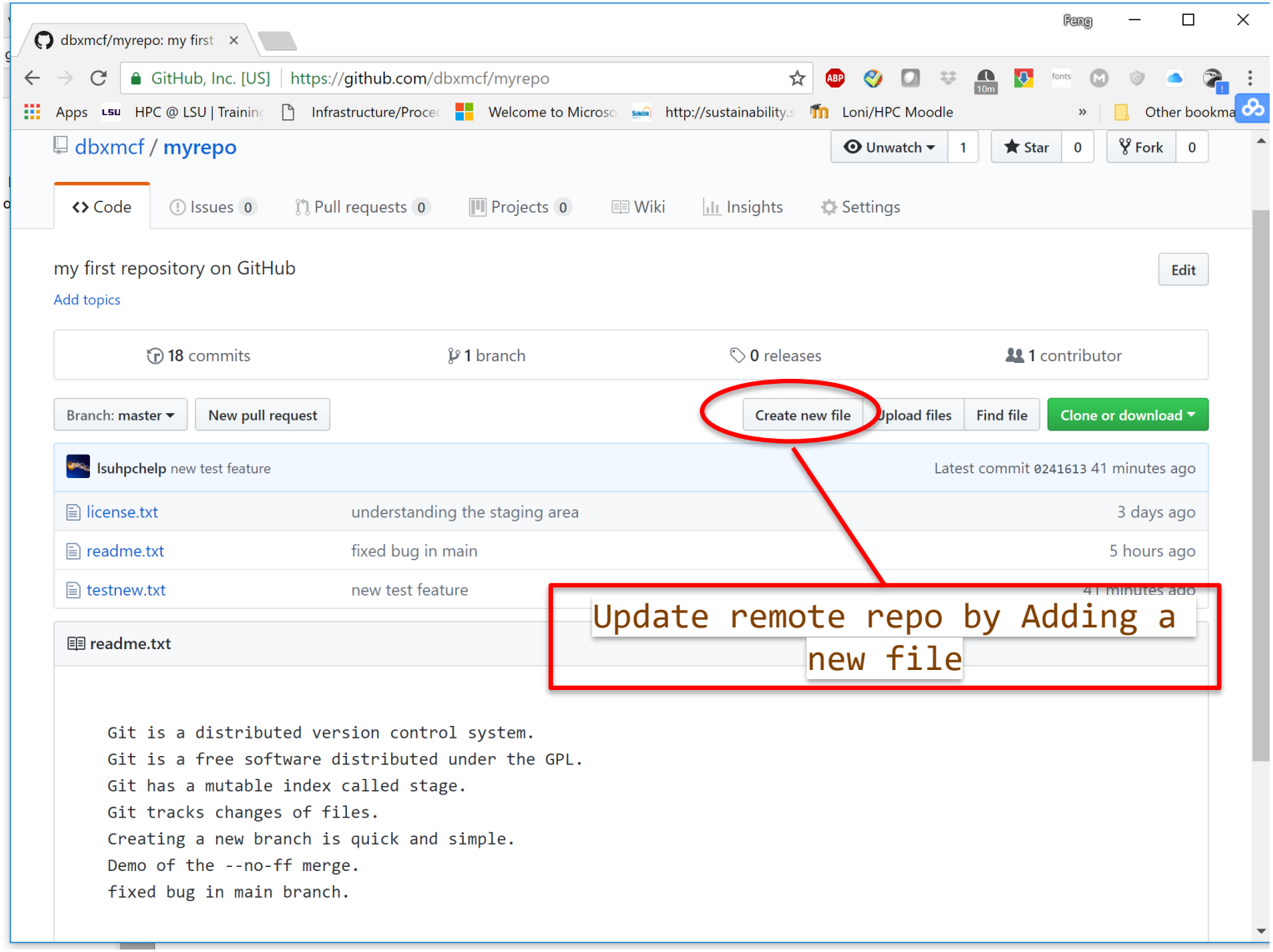
Push your current repository to GitHub

- **Copy and paste the commands in the previous slides to push your local master branch to GitHub**

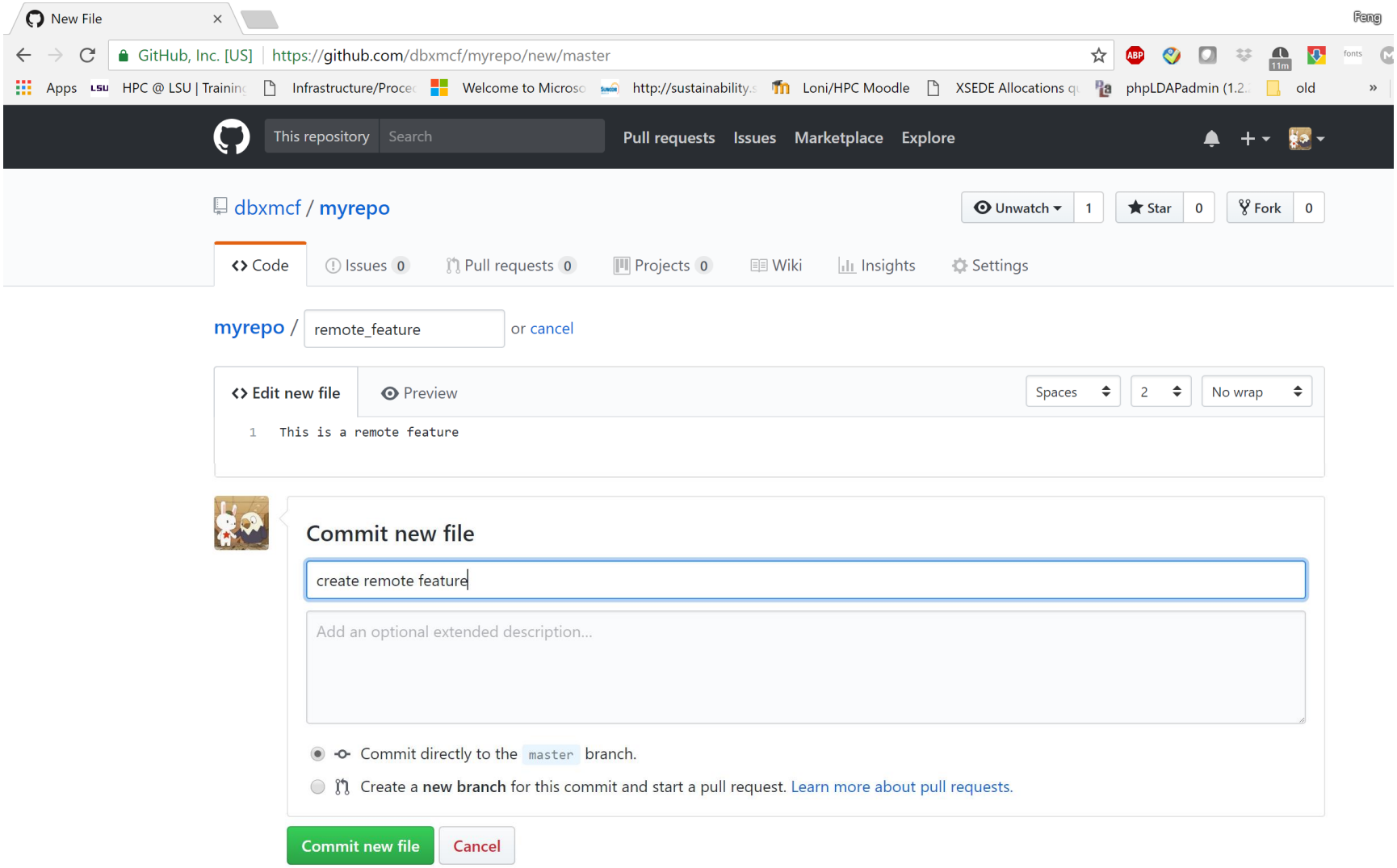
```
[fchen14@smic1 myrepo]$ git remote add origin https://github.com/dbxmcf/myrepo.git
[fchen14@smic1 myrepo]$ git push -u origin master
Username for 'https://github.com': dbxmcf
Password for 'https://dbxmcf@github.com':
Counting objects: 47, done.
Delta compression using up to 16 threads.
Compressing objects: 100% (42/42), done.
Writing objects: 100% (47/47), 4.19 KiB | 715.00 KiB/s, done.
Total 47 (delta 15), reused 0 (delta 0)
remote: Resolving deltas: 100% (15/15), done.
To https://github.com/dbxmcf/myrepo.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
[fchen14@smic1 myrepo]$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```



View your remote GitHub repository



Update remote repository



Updating your local copy of the remote branches

- **Use** `git fetch + git merge`
  - This is equivalent to "`git pull`"
  - `git pull = git fetch + git merge`

```
[fchen14@smic1 myrepo]$ git fetch
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/dbxmcf/myrepo
   0241613..c105f0b  master     -> origin/master
[fchen14@smic1 myrepo]$ git merge # merge the remote branch with local
Updating 0241613..c105f0b
Fast-forward
 remote_feature | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 remote_feature
```

Explore remote repository

- **Showing remote repositories**

```
[fchen14@smic1 myrepo]$ git remote -v
origin  https://github.com/dbxmcf/myrepo.git (fetch)
origin  https://github.com/dbxmcf/myrepo.git (push)
```

Summary

- **Why Git?**
- **Git locally**
  - Create repo
  - Working directory/Staging area/Repository
  - Manage the changes
- **Git branch**
  - How to create new branch
  - Merge branch
    - Fast-forward
    - Three way - Conflict
  - Branch management strategy
- **Git remote**
  - Push your local repo to the remote repo
  - Update your local repo from the remote repo

Version Control using Git

Question and Lab Session