Learn Git with GitKraken

Democritus University of Thrace
IEEE Student Branch

MLH
MAJOR LEAGUE HACKING

axosoft GitKraken

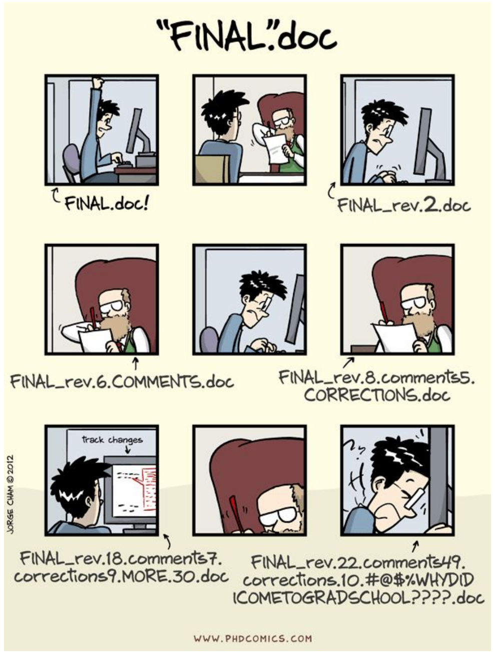
MLH localhost

how do you manage your coursework?

The nightmare of a university student:

- Multiple versions of the same document?
- Backing up working code?
- Sharing code in group projects?
- Emailing code to people?

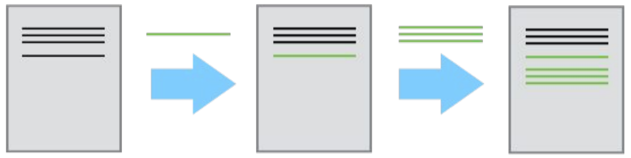
Now think about the thousands developers in Linux Kernel!!!


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

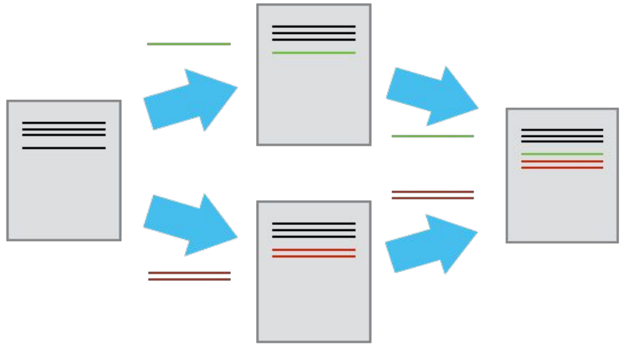
what is version control

Version control can help you

Start with a base version of the document and then record changes



Version control also allows many people to work in parallel:



Users can make independent sets of changes on the same document.

what is git

Git is a distributed version-control system for tracking changes in source code during software development

Created by Linus Torvalds, creator of Linux, in 2005

- Came out of Linux development community
- Designed to do version control on Linux kernel
- The most popular version control system
- Can track changes in any set of files

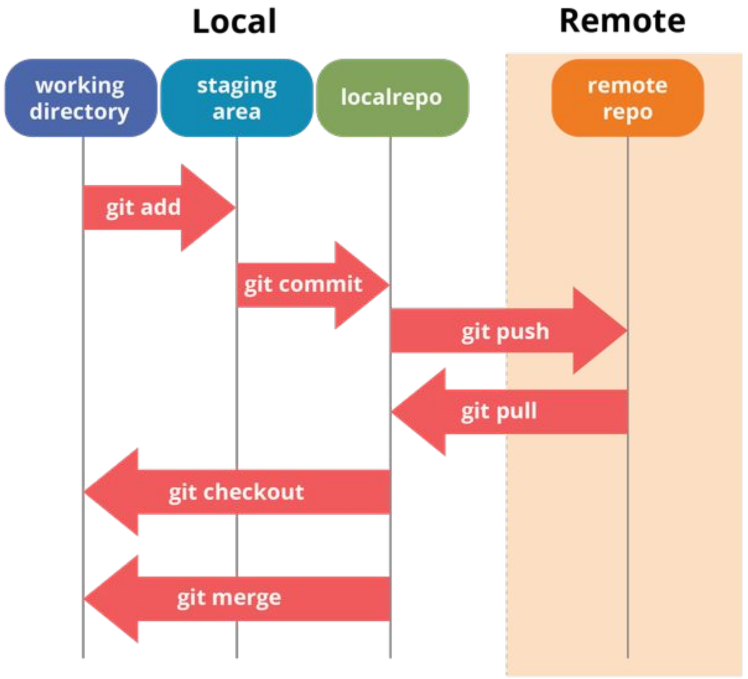
repositories

Files are kept in a **repository**

Repositories can be local or remote to the user

The user edits a copy called the working copy

Changes are **committed** to the repository when the user is finished making changes



installation (linux)

On a Linux distribution, check if git is already available on your machine

```
$ git --version
```

If not, you can install it by using the respective distribution’s package manager

for Ubuntu-Debian

```
$ sudo apt-get install git
```

for Fedora

```
$ sudo dnf install git
```

installation (windows)

Download git bash from https://gitforwindows.org/ and follow the installer

Make sure the following options are selected:

- Use the nano editor by default
- Git from the command line and also from 3rd-party software
- Checkout Windows-style, commit Unix-style line endings
- Use MinTTY(the default terminal of MSYS2)
- Enable file system caching/Enable Git Credential Manager

setting up

When we use Git on a computer for the first time, we need to configure a few things

The most important are the name and email address (use the one you have on GitHub)

```
$ git config --global user.name "John Doe"
$ git config --global user.email "john@example.com"
```

The flag --global tells Git to use these settings for every project, in your user account, on this computer

Check if the variables have been set correctly:

```
$ git config --list
```

git init

create a directory in Desktop folder for our work and then move into that directory

```
$ cd ~/Desktop
$ mkdir planets
$ cd planets
```

tell Git to make planets a repository – a place where Git can store versions of our files

```
$ git init
```

git init will create a repository that includes all subdirectories and their files

git status

Write a line in the mars.txt

```
$ nano mars.txt
Cold and dry, but everything is my favorite color.
```

Now check the file

```
$ cat mars.txt
```

We can check the status of the file.

```
$ git status
```

Untracked files message!

git add

We must tell Git to track a file

```
$ git add mars.txt
```

Git now knows to keep track of mars.txt, but it hasn’t recorded these changes as a commit yet

git commit

To get Git save a change as a commit we need one more command

```
$ git commit -m "Start notes on Mars as a base"
```

When we run git commit, Git takes everything we have told to save by using git add and stores a copy permanently inside the special .git directory

This permanent copy is called a commit (or revision), it has a shorter identifier. Your commit will have another identifier

git log

We can ask git to show us the project history using git log

```
$ git log
```

The git log command lists all commits made to a repository in reverse chronological order

Here we can also see the unique identifier of each commit

make changes

Let's add some more information in the file

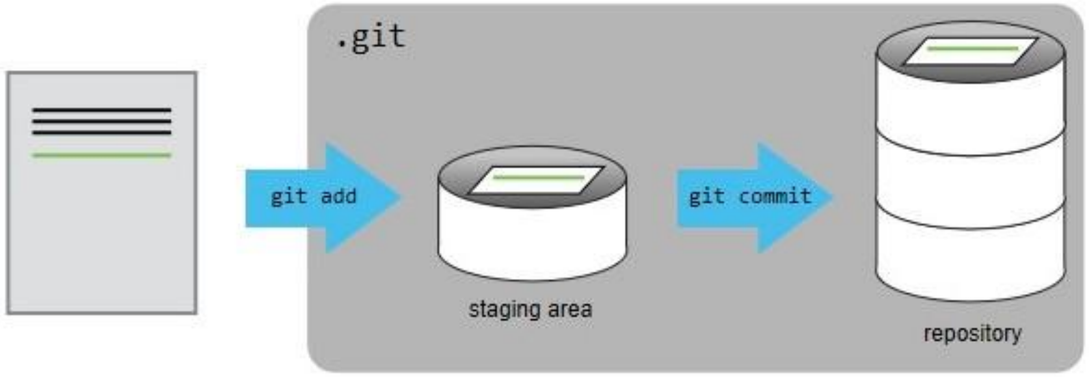
```
$ nano mars.txt
Mars has two moons: Deimos & Phobos!
$ cat mars.txt
```

Again, we must tell Git to keep track and save the file

```
$ git add mars.txt
$ git commit -m "Add info about Mars moons"
```

staging area

Git insists that we add files to the set we want to commit before actually committing anything. This allows us to commit our changes in **stages**.



To allow for this, Git has a special **staging area** where it keeps track of things that have been added to the current changeset but not yet committed.

git diff

Make some more changes

```
$ nano mars.txt
You have to appreciate the lack of humidity
$ cat mars.txt
```

Now we can see the differences between the current version and the previous commit of the file

```
$ git diff
```

git diff

What if we want to recover old versions of files?

We can refer to commits by their identifiers.
You can refer to the most recent commit of the working directory by using the identifier HEAD.

```
$ git diff HEAD mars.txt
```

You can refer to previous commits by using ~

```
$ git diff HEAD~1 mars.txt
```

git diff

```
$ git log
```

we can also refer to commits using their unique 40-character identifier

```
$ git diff f22b25e3233b4645dabd0d81e651fe074bd8e73b mars.txt
```

or fewer characters

```
$ git diff f22b25e mars.txt
```

git checkout

How can we restore older versions of our project?

git checkout: checks out an old version of a file
we recover the version of the file recorded in HEAD
use a commit identifier if you want to go back even further

```
$ git checkout f22b25e mars.txt
$ cat mars.txt
$ git status
```

git reset

What if you mess up and add some changes but you regret it?

git reset: unstages changes from the staging area

```
$ git reset <filename>
```

```
$ git reset .
```

unstages all staged files.
If you seriously mess up,

```
$ git reset --hard HEAD~3
```

It permanently deletes the last 3 commits and removes all changes from your working tree!

.gitignore

What if we want to tell git to exclude files from tracking?
editor files or intermediate files created during data analysis.

```
$ mkdir results
$ touch a.dat b.dat c.dat results/a.out results/b.out
```

Write everything you want git to ignore in a .gitignore file

```
$ nano .gitignore
*.dat
results/
```

Can you guess what this does?

.gitignore

```
$ git status
```

```
$ git add .gitignore
$ git commit -m "Ignore data files and the results folder."
```

other useful patterns:

```
*.dat
!final.dat
```

```
# ignore all .dat files
# except final.dat
```

```
results/data/position/gps
results/data/position/gps/*.dat
```

```
# all files in a certain folder
# all .dat files inside a certain folder
(paths are always relative to .gitignore)
```

to override the .gitignore rule

```
$ git add -f a.dat
```

git remote

There are many websites that host remote repositories for free, like GitHub, GitLab or Bitbucket

Open your GitHub account, create a new repository and copy the clone link

In order to connect your local repository with this remote repository:

```
$ git remote add origin https://github.com/user/repo.git
```

In order to see the changes

```
$ git remote -v
```

git push - git clone - git pull

In order to update the remote repository with yours

commit: adds changes to the local repository

push:transfers the last commit(s) to a remote server

```
$ git push origin master
```

clone: get a copy of an existing remote Git repository

```
$ git clone https://github.com/user/repo.git
```

pull: update your local repository with the contents of a remote

```
$ git pull origin master
```

resources

Version Control with Git - https://swcarpentry.github.io/git-novice/