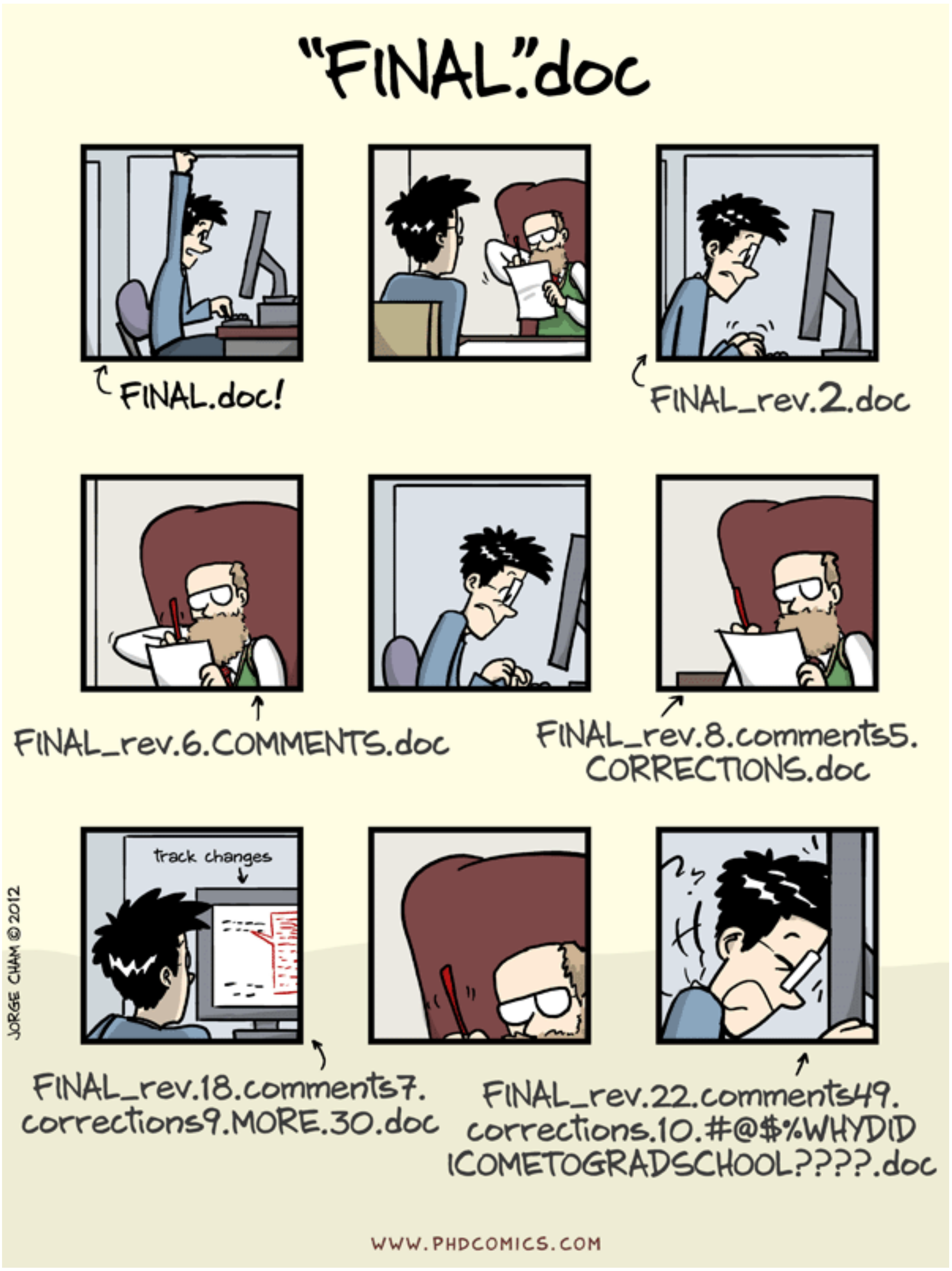
GIT BASICS

A **version control system** (VCS) manages files and tracks changes to those files. It stores the complete **history** and allows you to recover files at any stage of the history (think "undo" to the beginning). Any serious software project uses version control but its uses go beyond software, e.g., documents and data can also be version controlled.



A set of changes is called a **commit**. It typically contains changes to multiple files. It also contains a timestamp, information about the user who made the changes, and a **commit message** that explains the changes. The history consists of all the commits. The VCS stores the history in a storage area called a **repository**.

Configuring Git

The first time you use `git` you need to tell it who you are: this information will be the user information in the commit history.

Personalization

Set your name and email.

Use a name and email address that can appear in the clear in the world because you will also use it to sign up and access the GitHub. During this school we will use Slack and GitHub all the time:

- Homework will be submitted to *slack* (only visible to you and the instructors).
- Codes and Hands-on sessions will be submitted to *private* repositories only visible to you, and the instructors.
- The final project can be submitted to a *public* repository, visible to everyone in the world.

Set the name and email associated with your commits (*Use your own name and email address!*):

```
roshan@roshan:~$ git config --global user.name "Roshan Shrestha"
roshan@roshan:~$ git config --global user.email
"roshanpra@gmail.com"
```

`git` basic command syntax

Git commands always follow the pattern

```
   git <verb> [options] [arguments ...]
```

In particular, `--help` is always an option. Also try `git help` and `git help tutorial`.

Creating a repository

A repository starts from a directory with files.

Create your `SOSC` directory

```
roshan@roshan:~$ mkdir SOSC
roshan@roshan:~$ cd SOSC
roshan@roshan:~/SOSC$ touch file1.py file2.py
roshan@roshan:~/SOSC$ mkdir Project
roshan@roshan:~/SOSC$ ls
```

Turn the `SOSC` directory into **repository** with the `git init` command:

```
roshan@roshan:~$ cd ~/SOSC
roshan@roshan:~/SOSC$ git init
```

A new hidden directory `~/SOSC/.git/` appeared. This is your actual repository (or database) where Git stores all its information. **Do not change anything in this directory** (unless you really know what you are doing) and **do not delete the `.git`** directory. I Now try the (possibly) most-used git command:

```
roshan@roshan:~/SOSC$ git status
```

which gives

`On branch master

No commits yet

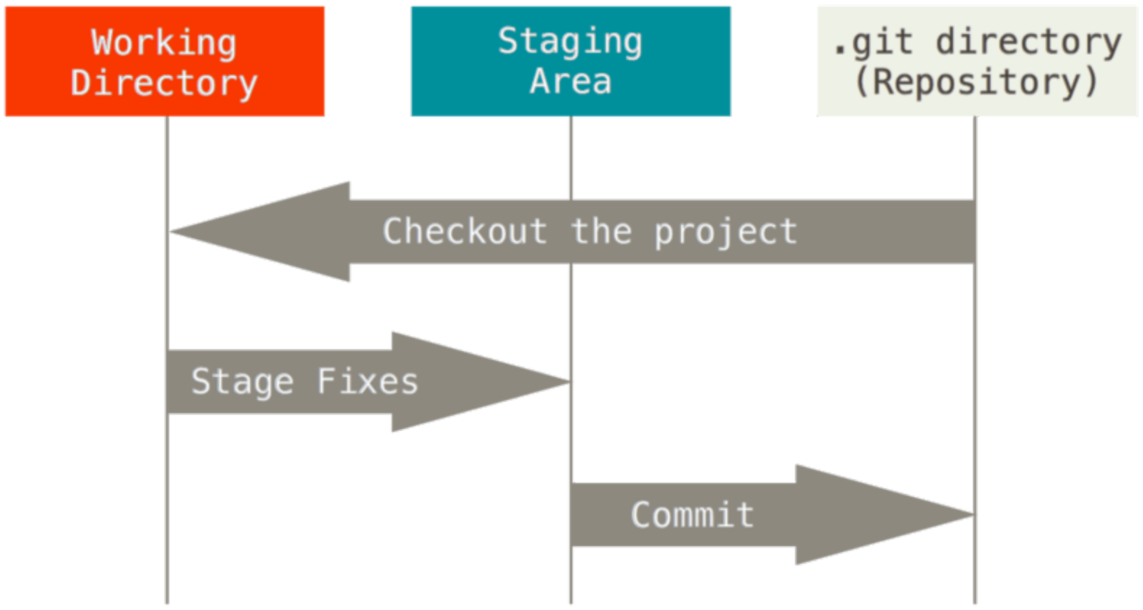
Untracked files: (use "git add ..." to include in what will be committed) file1.py file2.py

nothing added to commit but untracked files present (use "git add" to track) `

The three states of Git

For Git, a file can reside in one of *three states*

- **Modified** means that you have changed the file but have not committed it to your database yet; the file is in the **working directory**.
- **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot; it lives in the **staging area**.
- **Committed** means that the data is safely stored in your local database (the **git repository** in the `.git` directory).



The *basic Git workflow*:

1. modify files in working directory
2. selectively stage changes that you want to include in your next commit (adds only those files to the staging area)
3. commit your changes (takes files from the staging area and stores them permanently in your Git repository)

Working with remote repositories

Repositories can live in your local file system or remotely on another server ("in the cloud"). Git provides a way to synchronize local and remote repositories.

There's nothing special about remote repositories. They do not really differ from your local repository. Git is a *distributed version control system* and any repository holds all the relevant history.

There exist many services that host remote repositories "in the cloud". One of them is GitHub.

Initializing from remote: `git clone`

To initialize a local repository from a remote source we use `git clone` :

```
git clone URL
```

For instance, let's clone the class repository where code and data will be posted by the instructor:

```
cd ~
git clone https://github.com/sosc-npl/winter-spring-
school21.git
cd winter-spring-school21
```

Note:

- The local name of your repository can be any name you like but it is customary (and less confusing) if you go with the default, which is the remote repository name without the ".git" suffix, i.e., *winter-spring-school21* for us.
- For all other `git` commands you *must* be *inside* your local repository:

Getting changes (reading): `git pull`

Update your local repository with any "upstream" changes by **pulling** from the remote repository:

```
git pull
```

Updating the remote (writing): `git push`

If one has write access to a remote repository then one can also update the remote repository with local changes (commits) by **pushing** all commits that the local repository has:

```
git push
```

(This will *not* work for *winter-spring-school21* because only instructors have write access, but below, I will tell you how to set-up and use your repository.

Set up your own GitHub repositories

1. Go to https://github.com and create a new account. It is free. Remember your GitHub **username** and the **password**.
2. Create a new repository *SOSC* (do not initialize it with a README or other file)
3. Note the repository URL https://github.com/USERNAME/SOSC.git
4. Add the remote repository to your local repository `~/SOSC` (replace *USERNAME* with your GitHub username):

   ```
   git remote add origin https://github.com/USERNAME/SOSC.git
   ```

   We named the remote "origin", which is a common choice for the main repository. You can have many different remotes, just give them different names. You can list them with

   ```
   git remote -v
   ```
5. Traditionally, git used the name "master" for the main branch of the work. However, this name is falling rapidly out of favor so we rename our branch to "main":

   ```
   git branch -M main
   ```

   (You only have to do this once.)

6. **push** your local history to the remote repository:

   ```
   git push --set-upstream origin main
   ```

   ```
   git push -u origin main
   ```

   You need to enter your username and password.

   You only need the `--set-upstream origin main` (or `-u origin main`) for the first time (it tells git which "branches" to associate with each other in local and remote). All further push operations will then simply be

   ```
   git push
   ```

   Look at the web interface at https://github.com/USERNAME/SOSC and see your changes appear.

For the rest of the semester, commit what you did during each class session to the `~/SOSC` repository on your laptop (and also push to your GitHub repository as a backup).

Note: Your GitHub repository is for in-class work.

Winter Spring School 2021 Workflow: Resources and Workspace

Code, notebooks, and files are being made available on GitHub in the **winter-spring-school21** repository Winter-Spring School 2021. For the rest of the school it will be assumed that you have a `~/winter-spring-school21` repository that you can update with a simple `git pull` as soon as new material is posted.

If you have not done so already, set up `~/winter-spring-school21` :

```
cd ~
git clone https://github.com/sosc-npl/winter-spring-school21.git
```

You should also have your own **workspace** repository `~/SOSC` set up(set-up-your-own-github-repositories).

Overview

When we start a new lesson you will typically go through the following steps to get code and data files.

1. pull latest changes from winter-spring-school21
2. copy the new directories and files into your workspace SOSC
3. work in your workspace
4. commit changes in your workspace
5. (optional) push changes in your workspace to your GitHub repository

Update resources ("pull resources")

Update when your instructors changed anything on the remote:

```
cd ~/winter-spring-school21
git pull
```

Check what's new

```
ls -lt
```

(newest at top)

Copy new files and directories to workspace ("copy resources")

You should not be changing anything inside `~/SOSC` (because the next `git pull` might try to change something that you did but instead *copy* whatever you want to change to your own work directory.

For example:

```
cp -r ~/winter-spring-school21 /SOSC
cd ~/SOSC/winter-spring-school21
```

Work, commit, push

Then work on the files **in the workspace**.

When you have changes that you want to track, `git add / commit / push` :

```
In [ ]:
```