Contents

Preamble

When you see JoeSchmoe, replace it with your username. This is a placeholder.

Jargon

Terminal or command line

A text-based interface which allows you to run programs installed on your computer.

GitHub

A site which uses Git.

Git

A software version control system that is history-based.

Repository, or repo for short:

A place where the **history** of all the work you (or others) have done is stored.

Cloning

Cloning a repository is akin to downloading it, but with all of the history intact.

It is ***not*** the same as downloading a ZIP file of the repository. This will destroy all history within the repository and leave you with a snapshot of the file system.

Remote:

A repository that resides on a URL, such as this url.

When you clone a repository, by default, you have one remote called `origin`.

This points to the URL from which you cloned the repository.

Forking

Forking a repository is making a copy of a repository at a specific point in its history.

This allows you to freely make changes that you can upload to your fork's remote.

Branch:

A branch is essentially an independent line of development.

This can be useful to segregate development versus release versions of software.

By default, git repositories start with one branch and it is called `master`.

We do not require you to use multiple branches or fully understand branches, but the technical description is available here.

Pull request:

A request associated with the differences between two repositories for another person to accept, modify, or deny those differences.

For example, If I changed your CSV file to remove some bad data, but I didn't know if you wanted me to do that, or feared our changes would conflict, I should make a **pull request** after **committing** those changes.

Staging

Preparing files, line additions, or deletions to be committed.

This is done through a command called `git add`.

For example, if you had a file called `coolData.csv`, you could stage it by typing `git add coolData.csv`.

Commit

A record of a single unit of work. Contains a title (commit message) and an optional extended description.

Commits are done by one person and can include any combination of: - Adding files - Adding lines to a file - Deleting files - Deleting lines from a file

This is done through a command called `git commit`.

With our previous staging step, we can commit that file by typing `git commit -m "Added cool data to CSV."`

Pushing

Taking local commits and uploading them to a remote.

This can be your own remote, the team remote, etc.

The syntax for pushing is as follows:

```
git push <remote> <branch>
```

If you're used to typing `git push`, it is shorthand for `git push origin master`.

To push to our team remote (assuming you've added the remote), the command is:

```
git push team master
```

Pulling

Asking a remote for commits that you do not have, and incorporating them into your local repository.

The syntax for pulling is as follows:

```
git pull <remote> <branch>
```

If you're used to typing `git pull`, it is shorthand for `git pull origin master`.

To pull from our team remote (assuming you've added the remote), the command is:

```
git pull team master
```

Tools

You will need:

- A Git client
- A GitHub account

Optional tools

- A graphical Git client. I (Henry) use SourceTree and GitKraken and find them very visually useful.

  The only shortfall of GitHub Desktop that I know of is that you cannot deal with remotes other than `master`, but *we will be dealing with multiple remotes.*

Our workflow

Because we have chosen the very standard "fork and pull request" as a basis for our workflow, you cannot just push work directly to the team's repository.

Rather, we have you "fork" the team code repository (create a copy based off of it) and create a "pull request" if you have made changes that may conflict with others' work: i.e. commits that modify it, delete it, create merge conflicts, etc.

We have chosen to make some exceptions for the sake of time and convenience.

If you have work that you KNOW will not break anyone else's work, you are free to push your work to the team repository without making a pull request.

One-time setup

These steps are one-time setup only, and you will only need to add the `team` remote every time you freshly clone your personal repository.

Forking the team repository

Fork the team repository located at

`https://github.com/illinoistech-itm/2019-team-07f`

By going to the upper-right side of the page and clicking the [Fork] button.

We are doing this to give you a separate version of the team repository that you can work off of, without the fear of overwriting other people's work.

Cloning your personal repository

Under the list of forks:

`https://github.com/illinoistech-itm/2019-team-07f/network/members`

You (Joe Schmoe) should see `JoeSchmoe / 2019-team-07f`.

Click on `2019-team-07f` after `JoeSchmoe`. This will take you to your fork of the team repository.

Then, copy the URL.

It should be:

`https://github.com/JoeSchmoe/2019-team-07f/`

In terminal, navigate to a folder that you would like the team repository to be stored in. For reference, mine is `P:\GitHub\ITMT-430\2019-team-07f\`.

Then, run

```
git clone https://github.com/JoeSchmoe/2019-team-07f/
```

If all goes well, you should have

Adding the team repository as a remote

In the folder to which the Git repository was cloned, type:

```
git remote add team https://github.com/illinoistech-itm/2019-team-07f
```

That adds a reference to the team repository, which is now called `team`.

Your workflow

Doing work

When you have finished some meaningful unit of work, make a commit!

Say you've added a file, `JoeSchmoe-report.txt`, that's a finished report. That's great!

1. Stage the file with `git add JoeSchmoe-report.txt`.
2. Commit the staged work with `git commit -m "finish my report"`
3. Push the work to your repository with `git push origin master` aka `git push`.

If you wish to put your commits onto the team repository, keep reading.

Adding commits I've done to my repository

Stage and commit like normal, and just run `git push`.

Adding commits I've done to the team repository

Commits that I know won't conflict with others' commits

Feel free to push these by running `git push team master`.

Commits that might conflict with others' commits

Do not push these to the team repository, pretty please.

Pushing work that affects files others are working on can cause a thing called a merge conflict, for either you or the other person.

When you think that a commit might screw with someone's work, you can do the following:

- Ask them about the files and/or commits involved
- Open a pull request from the team repository's page (`New pull request` button) and ask them to review it

If they OK your dialogue or the pull request, go ahead and merge the pull request from the site and/or push to `team`.

Getting the team's commits

To get the team's work, type:

```
git pull team master
```

This asks the `team` remote for all commits that your local repository does not have, and attempts to incorporate them into your local repository.

Exercises

Now that you've read a bit (or too much) about Git and software version control, you can do some of these exercises to get a better feel for the workflow of our group.

Staging and committing

Inside the repository's folder, make a new R file, perhaps `cool-code.R`.

Before staging, run `git status`. Note that it shows up under `Untracked files:`.

To stage it, run `git add cool-code.R`.

Run `git status` again. Aha! It's staged!

Now, to commit. Run `git commit -m "add cool R code"` to commit your work.

Pushing to your repository

With our previous R file committed, run `git push`. It's that easy!

Feel free to look on `https://github.com/JoeSchmoe/2019-team-07f/` to see your changes.

Pushing to the team repository

Similar to the previous command, just run `git push team master`.

If you look at the team branch, at `https://github.com/JoeSchmoe/2019-team-07f/`, the commit and file should be there!

Tips and tricks

Git

When using git, there are a few useful commands you should know about.

- `git status` will tell you what files are staged, unstaged, and how many commits you have yet to push.
- `git diff <PATH1> [PATH2]` will show you the differences between two paths or files.

Opening Terminal

Windows

In Windows, you can open the terminal in a plethora of ways.

Anywhere

- `WIN-R`, type `cmd`, hit `ENTER`.
- `WIN`, type `Command Prompt`, hit `ENTER`.
- `CTRL-SHIFT-ESC`, `File > New Task (Run...)`, type `cmd`, hit `ENTER`.

Windows Explorer

This is highly preferable to `cd`ing around endlessly, and much faster.

- In Windows Explorer, Tap `ALT`, tap `SHIFT-F`, tap `W`, and hit `ENTER`.
- In a folder, `SHIFT-RMB` on the body of a folder. Select `Open command window here`.

Mac

Anywhere

- Open Spotlight with `COMMAND-SPACE`.

  Start typing in `terminal` and press `ENTER` when it is selected.

Finder

Unfortunately, you can only do this with some moderate fiddling.

Linux

`CTRL-ALT-L` and the menu bar (File, Edit, View, etc.) are good places to start.

Other than that, you're on your own! Not covering this for Linux as DMs vary quite wildly.

I recommend `tmux` and Google.

What's in my current directory?

Use `dir` on Windows and `ls` on *nix. (OSX is *nix)

Changing directories

Changing directories is done on almost all platforms by using the `cd` command.

This command takes one argument, and it is a path.

For example:

Consider the following directory structure:

```
C:\IMAGES
+---puppies
+---snakes
\---turtles
```

`cd C:\images\turtles` Takes me to my turtle pictures folder, no matter where I am.

`cd ..` Takes me back to `C:\images\`.

`cd snakes` would take me to `C:\images\snakes` because I ran it while I was at `C:\images`.

Special Paths

- `.` means "the current directory".
- `..` means "the directory that contains our directory".

  You can chain this. For example, `..\..` means "two directories above us".
- `/` (on *nix) means "the root directory, which contains all others".

Relative vs Absolute paths

A path can be relative or absolute.

If my current directory is `C:\images\puppies`, and I try to `cd cats`, I would be attempting to go to `C:\images\puppies\cats`.

Good god, this filename/path is long!

Good news! In (most) terminals, paths and filenames can be auto-completed and sometimes suggested.

Just mash `TAB`!

If you're unsure what I mean, just start typing your path and halfway through, start randomly pressing `TAB`.