Introduction to Git

What is Git?

Git is a version control system, i.e., a software that allows you to manage different versions of files in a repository.

The repository is a directory that is managed by git, and it can be on your computer or on a remote server.

Git can be used to: * manage and access different versions of files * work on the same set of (text) files with several (or a lot of) people and solve the resulting conflicts that may occur * do many complex things which are specific to software development that we do not care about today

Git can help you to solve to the following problems: * You messed up your R script today when working on it, and you want to get the version from yesterday back. * You want to know the settings you used when computing the data for paper XY, but you have changed the script in the meantime. * You want to see the changes introduced to a script since last month. * Both you and a colleague made changes to the same script. You want to merge them to create a new version based on both change sets. * You are working on a script over time, and you want colleagues to be able to access the latest version (without sending emails to them after every change) * You want colleagues or others to be able to see your code, and maybe even allow them to directly make or suggest changes (which you can accept or refuse)

Alternatives to Git include Mercurial, Darcs, and Apache Subversion.

Important Git Terms (and commands)

For local use:

- repository or repo: a git-managed directory containing all files of a project
- commit: the act of saving changes to a repo. Each commit is given a commit hash: a (more or less) random string that identifies it
- tag: a name assigned to a commit, so that it is easier to find it in the history (example: "version_used_for_nature_paper")

When using remote repos:

- origin: the original version of a repo (when your repo is a copy of another repo, that other repo is its origin)
- clone: the act of creating a copy of a repo
- merge: combining different versions of files.
- conflict: occurs when two incompatible versions of a file need to be merged. Note: Git can auto-merge non-conflicting changes, and will detect conflicting changes and ask the user to fix them.
- pull: refresh your local repo by merging in changes from its origin
- push: the act of merging the changes in your local repo back into its origin

Presentation

Let's get started

Ensure your git installation works

First make sure you have downloaded and installed git. Just answer all questions with the default setting.

- Start a command line:
  - Windows: start Programs / Git / Git Bash
  - MacOS: in Finder, click Applications / Terminal
  - Linux: in the Start Menu, click System Tools / xterm

Now type `git --version`. You should see version information for git. If you get an error like `Command not found` instead, you either have not installed git or have not yet added it to your PATH environment variable. You will need to fix this before proceeding.

Perform first-time setup

Note that these are not credentials for some remote server, they are just something to identify you when you add something to a repo. You can put whatever you want.

```
git config --global user.email "you@example.com"
```

```
git config --global user.name "Your Name"
```

Create your first repo and add a file

Let's create a new repository:

```
pwd
mkdir myrepo
cd myrepo
git init
```

Now let us create a file named README.md in the repo and add it. You can use a text editor to create a file, or just use the command line:

```
touch README.md
git add README.md
git commit -m "Add empty readme file"
```

It's time to make a change. Edit the file using a text editor and add a line, e.g., "# My public R project". Make sure to save the file. Now let us add the changed file.

```
git status
git add README.md
git commit -m "Add basic info to readme file"
git status
```

We now have a second version of the file in the repo. Let's view the commit history:

```
git log --oneline
```

Note the commit hashes which are displayed. Let us jump back to an earlier version of the repo, identified by a commit hash:

```
git checkout <hash>
```

And return to the latest version again:

```
git checkout master
```

Alternatives to the command line

There are many graphical user interfaces for git, and many advanced text editors (like Atom) and all major integrated development environments (IDEs, like rstudio, netbeans, pycharm, VSCode,...) have support for git.

Some general tips

Things you should do:

- Separate data and code
- Commit often (several times per day, after every major change). It does not hurt and it ensures you always have the latest version available, should you need to continue working on another machine.
- Write scripts in a machine-independent way

Things you should not do:

- Store credentials (passwords) in any Git repository
- Store copyrighted material you do not own, personal data, or patient data of any kind in a public Git repository
- Use git as a replacement for backups
- Check-in large data files
- Check-in files which can be easily generated from other files in the repo (e.g., you should add the Rmd file, but not the PDF, doc or html files which you can export based on the Rmd file)

Getting a server

Important: Before creating a repository (public or private), you have to make sure that you are allowed to store your data on that server. Typically you are not allowed to store patient data or metadata on servers outside of your organization. If you need to version such data, you should talk to your IT department to get an internal git server.

For data that can be stored on external servers, there are hosting platforms like github, gitlab or bitbucket which provide git hosting combined with a convenient web interface to browse and download files and revisions. These platforms often include other tools as well, e.g., project management and bug tracking software, or a wiki.

Most platforms offer repos for free if you are working alone or in a small team. The only service I know of that offers private repos for free is bitbucket, but I recommend to check for yourself.

Using a remote server

Once you have a server, you can:

- create a repository on it: There should be a web interface, just login and follow instructions to create a repo. Typically, the new repo will be created at an URL like https://serviceprovider.com/you_username/repo. In many cases, it contains a README file explaining how to use it.
- clone the repo on your local machine: `git clone https://serviceprovider.com/you_username/repo`
- Work with the repo as explained above (add files, modify them, commit as often as you want locally)
- Once you feel like syncing your local repo with the remote repo on the server, push your changes: `git push`. This will ask for your username and password for serviceprovider.com. Once it is done, you should see the changes in the web interface.

Before you start working on another machine, make sure to refresh your local repo from the remote server: `git pull`

More info on git

- The official git documentation website: great documentation, tutorials, videos, ...
- the slides used for the course, from courses.cs.washington.edu
    - License and credits for the linked presentation: Portions of the CSE403 web may be reprinted or adapted for academic nonprofit purposes, providing the source is accurately quoted and duly creditied. The CSE403 Web: © 1993-2019, Department of Computer Science and Engineering, Univerity of Washington

Git workflows for science

- For data analysis: work in a private repository, create a new public repo with assorted files (and without the history) and publish it once the paper is published
- Use one repo per group to share instructions / knowledge and scripts

Getting the latest version of this document

You can get the latest version via git:

```
git clone https://github.com/dfsp-spirit/reproducible_science.git
```

If you already have a copy of the repo, just refresh it from time to time:

```
cd reproducible_science/
git pull
```