Stat 405

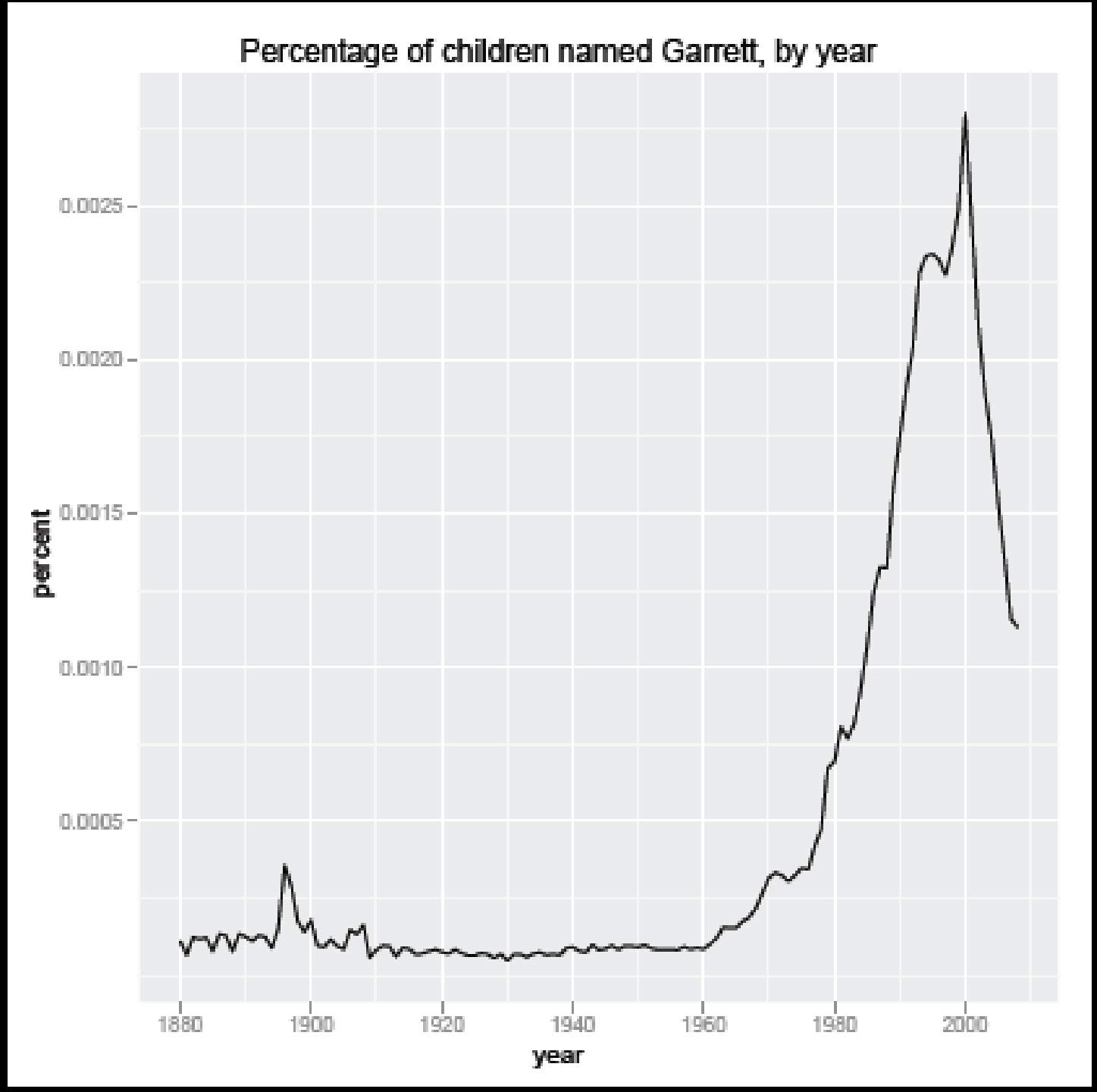
Introduction to git

Garrett Grolemund

1. Why git?
2. Setting up git
3. Using git
4. More reasons for git

Why git?

- Download git-project.r and git-template.tex
- Download baby-names.csv
- What does git-project.r do?


Percentage of children named Garrett, by year
percent
0.0025
0.0020
0.0015
0.0010
0.0005
1880
1900
1920
1940
1960
1980
2000
year

Your turn

- Use the git-project.r. code to make a graph that shows the percentage of children with your name by year. Make the graph, but don’t save the changes to git-project.r.

- Hint: you only need to change two words, three if you are female.

Group projects

Working on our own is simple. But what if we wish to pool our efforts?

- Pair up with a partner

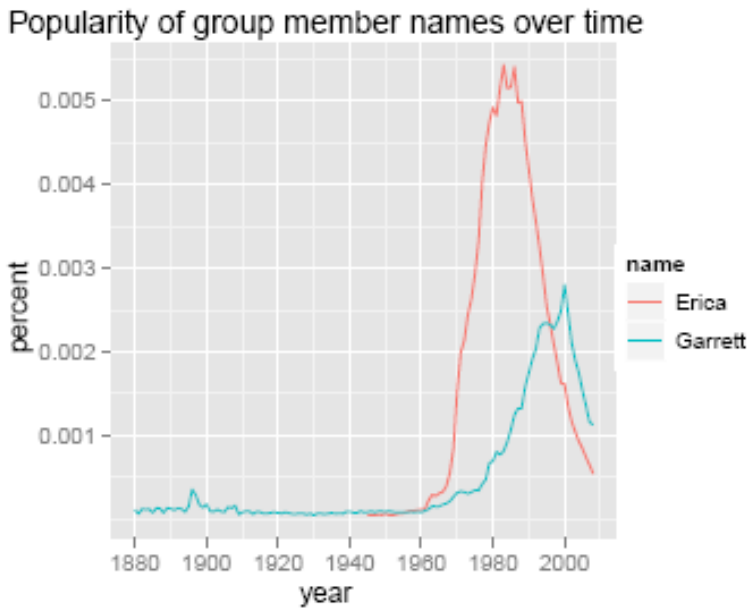
- Our goal: to create a one page report about your two names. Include a graph that compares the popularity of each name over time.

Popularity of "Garrett" vs. "Erica" as children's names

Garrett Grolemund

October 16, 2009

Below I have plotted the popularity of both my name and my sister's name over the past 129 years. Popularity is measured by the percent of newborn babies (of the appropriate gender) that were given the name each year.



As you can see, my sister's name used to be much more popular than mine. But who's laughing now? For the past 13 years "Garrett" has been more popular than "Erica."

Caveat: you must each work from your own computer.

Brainstorm

How would you exchange information? How many people could work on the code at once? How would you back up your work? What difficulties might you encounter?

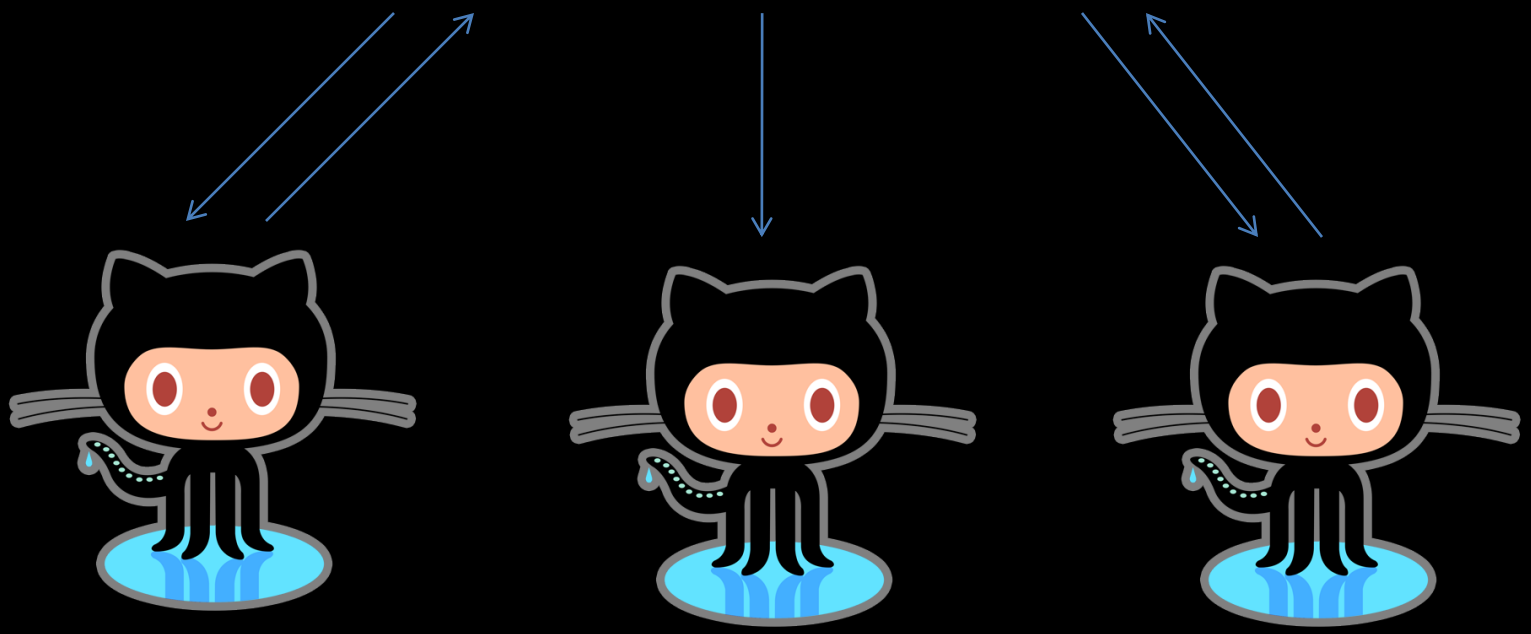
Pair up and brainstorm for 2 minutes

Git

Git is a program that makes project management easier. Github is a website that makes using git easier.

What does git do?

1. Git backs up the entire project and the entire history of the project. You can undo mistakes very easily.
2. Git allows multiple people to work on the same code at once without overwriting each other's work.
3. Github allows you to keep all of your files in one central, publicly accessible location.



Think of git as a library where your files live. The library stores the most up to date version of the files as well as all previous versions. Multiple users can check out the same file at the same time. Whenever you return the file, the library writes a new most up to date version that incorporates your changes.

Github

Github provides an online location for this "library." It is a website that hosts project files for git users. It also offers its own project management tools, and it can be used to share your project with the public.

www.github.com

A sample repository on github:

www.github.com/hadley/data-housing-crisis

Setting up Git

Configuring git

After you install git, open the terminal window and run the following commands. These will make working with git easier.

```
git config --global user.name 'John Doe'
git config --global user.email jd@example.com
git config --global color.ui auto
git config --global color.branch auto
git config --global color.status auto
git config --global color.interactive auto
git config --global branch.autosetupmerge true
```

Creating ssh key

To register for a github account, you will need to provide an ssh key. This is how github will recognize the messages your computer sends.

Instructions for generating an ssh key for Macs, Linux, and Windows are each on the handout.

Your turn

Register for a free github account at www.github.com. You will need to create a ssh key to use. Run

```
ssh git@github.com
```

at the command prompt to ensure the ssh key is working. You should get the message:

Hi username! You've successfully authenticated, but GitHub does not provide shell access

You have now:

1. Installed git (before class)
2. Created a repository

Now we want to put our work into the repository and begin collaborating.

Git init

Git does not automatically track every file on your computer. You must tell it which files to monitor.

To do this, open a terminal window and navigate to the folder you wish git to track.

```
cd <filepath>
```

Git init

From within this folder run the command:

```
git init
```

This “inititalizes” the folder. Git will now pay attention to the folder and its contents. You only have to run git init once per the lifetime of a folder.

Linking to github

To link the folder to your github repository, run the command:

```
git remote add origin
    git@github.com:<username>/
    <repositoryname>.git
```

(All one line) Your username and repository name are in the address bar of the github webpage for your repository. Like `git init`, this command only needs to be run once. Note: don't include the <> symbols.

Your turn

Go to your dashboard page on github. Click “create your first repository” and follow the directions. Github will give you a short script to run. You’ll recognize `git init` and `git remote` in the script.

Create a new folder on your computer named “name-project”.

Navigate to that folder and then run the script github gave you.

Adding to the repository

Now that we have a repository, we'd like to put some files into it.

Sending files to a repository is a three step process:

1. First we tell git which files we want to send
2. We package these files up with a message
3. We send the package to the repository

Git add

We tell git which files to send with `git add`. To add files one by one, use

```
git add <filename>
```

Or, to add all of the files in the folder at once, run:

```
git add .
```

Git commit –m""

Once we've selected all of the files to update, we need to combine them into a package to send to github.

These "packages" are called commits and contain two parts:

1. The files to add to the repository
2. A short message describing the files

Git commit –m""

To build a new package out of the files that have been added, run

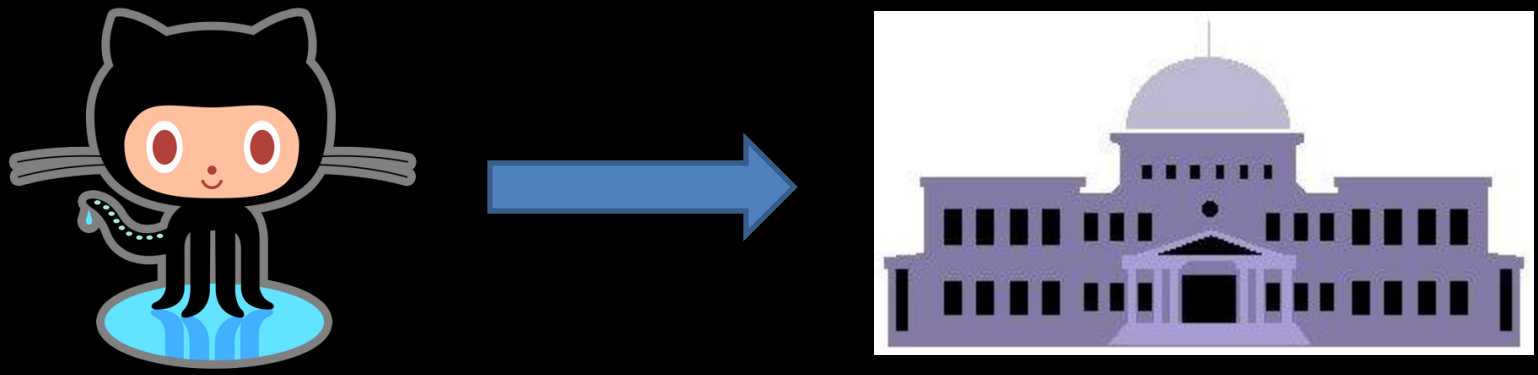
```
git commit –m"your message here"
```

The message will appear on github and will help you navigate back through points in the project's history.

It will also help you and your coworkers understand what you included in the package.

Git push

Now we just need to send our neatly packaged files to the github repository.



Doing this is very simple. Just run:

```
git push
```

Your turn

Move the original git-project.r and git-template.tex files into the folder linked to the git repository.

Use `git add`, `git commit -m""`, and `git push` to send the files to your github repository.

Git clone

At some point you may want to collaborate with a researcher who already has files in a github repository.

With one command you can 1.) copy these files to your computer, 2.) link the files to the github repository, and 3.) tell git to start monitoring them. To do so, create a folder and from within it run:

Git clone

```
git clone
  git@github.com:<username>/<repo
  sitory>.git
```

(All one line). The address that you'll need after "clone" is the same address that appears on the researcher's github repository page in the first box down. Like `init` and `remote`, this command only needs to be run once per folder.

Your turn

Decide with your partner which repository to use for today's project (theirs or yours).

Repository owner: click the Admin tab on your github repository webpage. Add your partner as a Collaborator.

Collaborator: clone the repository into a folder on your computer.

Using Git

Now that everything is set up, using git is easy. With git, you will always follow the same process:

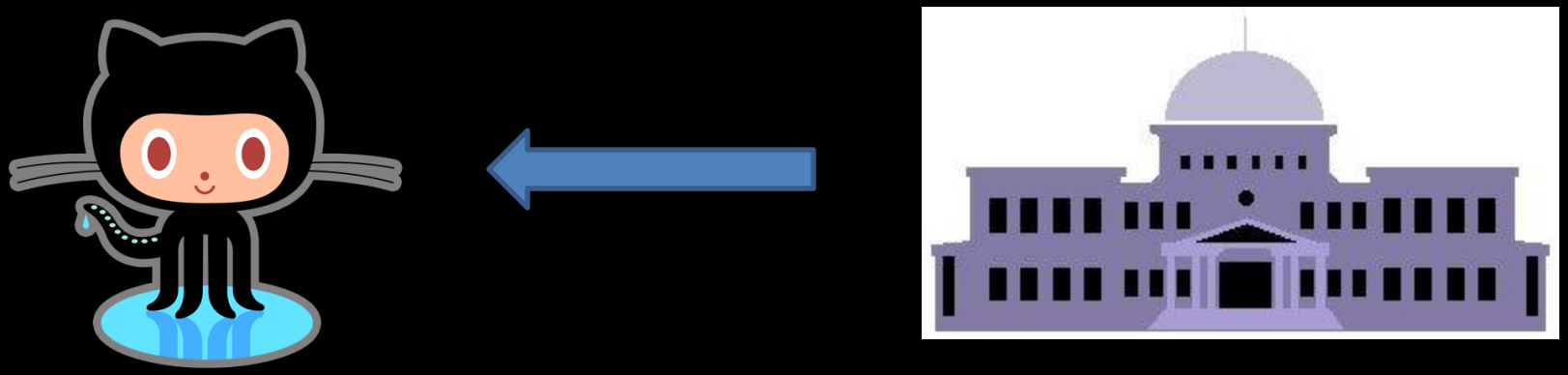
1. Checkout the most current version of the project from the repository (`git pull`)
2. Work on the files and save your changes
3. Review what changes you have made (`git status` and `git diff`)

4. Tell git which changed files to update the repository with (`git add`)
5. Package those files up (`git commit -m""`)
6. Send those files to github (`git push`)

Github will automatically update the project with the changes you made.

Git pull

Git pull does the opposite of git push. It brings updated files from the repository to your computer.



In both cases, out of date files are automatically replaced with their updated versions.

Git pull

When working with a group, always pull the most up to date version of the project before beginning work.

- If someone has worked on the project recently, the files on your computer may be obsolete.

To pull, navigate to the folder you want to work on and run:

```
git pull
```

Git pull

`git pull` won't work if you have "uncommitted" changes

- You must first save your changes with `git add` and `git commit` or remove them.
- Why? Git doesn't want to overwrite any important changes that you've made

Your turn

Use `git pull` to update your folder with the files in the repository that your group has chosen to use.

Open the new version of git-project.r and change the code so that it graphs your name. This time save the changes.

Git status and git diff

It’s a good idea to keep track of what you’ve been changing.

- `git status` shows you which files in your folder have been changed. It also shows you which files have been “added” for the next commit and which haven’t been added.
- `git diff` shows you the exact changes that you’ve made within each file

Git checkout

Sometimes you don’t want to save your changes. You can remove changed files with

```
git checkout <filename>
```

Or

```
git checkout .
```

To remove all uncommitted changed files.

This replaces the changed files with their old versions stored on github.

Git rm

Sometimes the change you wish to make involves deleting a file. You can delete a file with

```
git rm <filename>
```

If you manually delete a file, you manually add and then commit this change to the repository. Otherwise git will continue to look for a file that no longer exists. You can add the "removal" with the command:

```
git add -u
```

- In general its easier to use `git rm` to delete files that git is tracking.

Your turn

Look at the changes you've made to git-project.r. Add those changes with `git add` and use `git commit -m""` to package them with an appropriate message. Then push them to github with `git push`.

Uh oh. What happened?

The repository changed while you were away. We'll have to retrieve the new files with `git pull` and then fix any conflicts that have arisen.

Merging conflicts

If two people check out the same version of code and then both alter the same *line of code,* git won't know which version to keep.

Git will ask you to show it how to blend the two changes.

Merging conflicts

To resolve a conflict:

1. Run `git diff` Git will show you exactly which lines are conflicting.
2. Open the file and rewrite those lines however you prefer. You'll notice that git has put markers in the file to help you find the lines.
3. Use `git add` and `git commit -m"fixed conflict"` to repackage the file
4. Run `git push` again

Your turn

Fix your merge conflict. Rewrite the code to include both names. You can also now use the bottom portion of the code to create a graph that compares both names.

Dealing with mistakes – git reset

If you ever make a mistake and want to back up to a previously saved commit, you can use `git reset --hard`

But be careful: once you reset and then push, your project will return to where it was at the specified commit. ALL changes since then will be gone.

Git reset --hard

To return to the previous commit, run

```
git reset --hard
```

To go back exactly two commits, run

```
git reset --hard HEAD^
```

To go back to some ancient commit, look up the name of the commit on the commits tab on github, and run:

```
git reset --hard <commit name>
```

*Please ask Professor Wickham or me before doing this last one. There's probably a better way to achieve your goals.

.gitignore

.gitignore is a file that you can save inside of any git tracked folder. Git will ignore any file whose name you put into the .gitignore file.

This is useful is you don’t want to send large datasets or extra latex files to the repository.

Each subfolder can have its own .gitignore file

Git ignore

To make a .gitignore file, open a text editor, write the file, and then save it as ".gitignore" in the desired folder.

Note: Files whose names begin with "." are considered system files. As such they remain hidden when you look at a folder's contents. To see all of a folder's contents, including hidden files, type:

```
ls -a
```

A basic .gitignore file could contain the following items. This will prevent git from updating all of the excess latex files.

```
.DS_Store
..Rcheck
*.old
*.bbl
*.pdfsync
*.aux
*.bak
*.blg
*.mtc
*.log
*.mtc1
```

*.out
*.toc
*.maf
*.fdb_latexmk
*.lof
*.lot
Autosaved
*.dvdproj
*.synctex.gz
*.adx
*.and
*.idx
*.ilg
*.ind

Your turn

Work together to fill in the git-template.tex latex template. One person fill out the template, one person make the graph. Create a .gitignore file before you start filling your repository with excess latex files. Use git to send material to each other. Push your final product to the github repository.

Other benefits of git and github

- Reproducibility
- Organization
  - Tip: divide work and create a to do list with github issues tab
  - See who is contributing and who is not with the github impact graph
- Online backup
- Will be helpful (and required) for a future project

Thank you.

I hope you find git useful.

Git resources

Large collection of help files

http://github.com/guides/home

Git cheat sheet

http://zrusin.blogspot.com/2007/09/git-cheat-sheet.html

Git User's manual

http://www.kernel.org/pub/software/scm/git/docs/user-manual.html