

Version Control with Git and Github

Mikhail Dozmorov
Fall 2017

Reproducible Research

Good science is built on research that may be reproduced (or copied) by other researchers (and yourself!)

- If another researchers read your report, could they reproduce your work?
- Could you reproduce your own analysis after a year? two?
- Intro to version control: https://rpubs.com/marschmi/ear523_Feb02

Tips for Reproducible Research

- Document all you do and use your code as the documentation
- Make figures, tables, and statistics the result of scripts
- Keep an entire project in a single directory that is version controlled

3/54

Keeping History of Changes

(Jorge Cham,
<http://www.phdcomics.com>)

The Lame Method

- Multiple dated files with largely



4/54

Keeping History of Changes

Version control is the only reasonable way to keep track of changes in code, manuscripts, presentations, and data analysis projects

- Backup of your entire project
- Promotes transparency
- Facilitates reproducibility
- Faster recovery from errors
- Easier collaborations

5/54

Why Version Control?

Version control is not strictly necessary for reproducible research, and it's admittedly some extra work (to learn and to use) in the short term, but the long term benefits are enormous

People are more resistant to version control than to any other tool, because of the short-term effort and the lack of recognition of the long-term benefits

Imagine that some aspect of your code has stopped working at somepoint. You know it was working in the past, but it's not working now. How easy is it to figure out where the problem was introduced?

6/54

What is Git?

- Git is an open-source distributed version control system
 - Developed by Linus Torvalds (developer of Linux)
- Distributed, distinct from centralized (subversion)
 - Authors can work asynchronously without being connected to a central server and synchronize their changes when possible
- Complete audit trail of changes, including authorship
- Freedom to explore new ideas without disturbing the main line of work
- Collaborate with elegance – on any file at any time

7/54

Advantages of version control:

- It's easy to set up
- Every copy of a Git repository is a full backup of a project and its history
- A few easy-to-remember commands are all you need for most day-to-day version control tasks

8/54

Setting up Git

Download and install Git, <https://git-scm.com/downloads>.

Do not install GUI client

- `man <command>`. `man git` - not always good
- `git --help [<command>]`

9/54

Setting up git

- `git config --global user.name "Your name here"`
- `git config --global user.email your_email@example.com`
- `git config --global color.ui "auto"`
- `git config core.fileMode false`
- `git config --list`

10/54

Configuring Text Editor

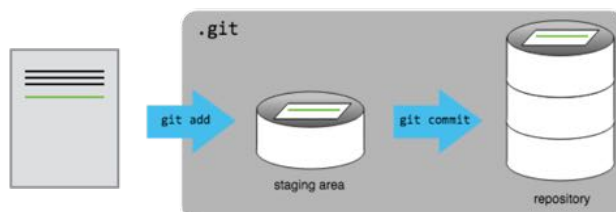
Editor	Configuration command
nano	<code>git config --global core.editor "nano -w"</code>
vim	<code>git config --global core.editor "vim"</code>
Text Wrangler	<code>git config --global core.editor "edit -w"</code>
Sublime Text (Mac)	<code>git config --global core.editor "subl -n -w"</code>
Sublime Text (Win)	<code>git config --global core.editor "'c:/program files/sublime text 2/sublime_text.exe' -w"</code>
Notepad++ (Win)	<code>git config --global core.editor "'c:/program files (x86)/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"</code>
Kate (Linux)	<code>git config --global core.editor "kate"</code>
Gedit (Linux)	<code>git config --global core.editor "gedit -s"</code>

11/54

Git Concepts

Two main concepts

1. commit - a recorded snapshot of differences you made to your project's files
 2. repository - the history of all your project's commits
- Files can be stored in a project's working directory (*which users see*) the staging area (*where the next commit is being built up*) and the local repository (*where revisions are permanently recorded*)



12/54

Starting a Git Repository

Exercise:

Make a folder. Check with `ls`

`git init` - initializes a repository

`.git` folder contains all Git info - remove it and all will be lost

`git status` - to see the status of your repository.

13/54

How the files travel

Lifecycle of files in Git repository

Untracked

Modified

Staged

Committed

14/54

Tracking Changes

- `git add` - puts files in the staging area
- `git commit` - saves the changes in the files on staging area to the local repository.
- Always write an informative commit message when committing changes, "-m" flag
- `git status` - shows the status of a repository



15/54

What to Add

New repository from scratch

The first file to create (and `add` and `commit`) is probably a `README.md` file, either as plain text or with Markdown, describing the project

A new repo from an existing project Say you've got an existing project that you want to start tracking with git. Go into the directory containing the project

- Type `git init` - initializes empty repository
- Type `git add <file> [<file> <file> ...]` - start tracking all the relevant files
- Type `git commit` - saves the snapshot of your current files

16/54

What NOT to Add

`git add`, `git add -u` – add all the changes (updates) in your files

- Don't include files that are derived from other files in the repository
- Be careful about committing binary files, or really big files.
- Git works best with text files, like code, you can see differences.
- Exercise: commit an image, overwrite it with another, `git diff`
- `git rm <file>` - removes a file from the current and future commits, but it remains in history/repository
- The `.gitignore` file tells Git what files to ignore. `git add -f` forces adding

17/54

Ignoring Unnecessary Files

- The various files in your project directory that you may not want to track will show up as such in `git status`
- Unnecessary files should be indicated in a `.gitignore` file
- Each subdirectory can have its own `.gitignore` file

18/54

Ignoring Unnecessary Files

Also, you can have a global `gitignore`, such in your home directory, e.g. `~/.gitignore_global`, which contains:

```
*~  
  
.*~  
  
.DS_Store  
  
.Rhistory  
  
.Rdata  
  
.Rproj
```

You have to tell git about the global `.gitignore_global` file: `git config --global core.excludesfile ~/.gitignore_global`

19/54

When to Commit

- In the same way that it is wise to often save a document that you are working on, so too is it wise to save numerous revisions of your code
- More frequent commits increase the granularity of your “undo” button
- Good commits are atomic: they are the smallest change that remain meaningful

**COMMIT EARLY,
COMMIT OFTEN**

<http://blog.no-panic.at/2014/10/20/funny-initial-git-commit-messages/> <http://www.slideshare.net/rubentan/how-we-git-commit-policy-and-code-review>

20/54

When to Commit

1. One commit = one idea or one change (one feature/task/function to be fixed/added)
2. Make and test the change
3. Add and commit

21/54

Best Practices

- A good commit message usually contains a one-line description of the changes since the last commit and indicating their purpose
- Informative commit messages will serve you well someday, so make a habit of never committing changes without at least a one-sentence description

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJBLKDFJSOKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

<https://xkcd.com/1296/>

22/54

Anatomy of Git Commits

- Each commit is identified by a unique “name” - SHA-1 hashtag
- SHA-1 is an algorithm that takes some data and generates a unique string from it
- SHA-1 hashes are 40 characters long
- Different data will always produce different hashes
- The same data will produce exactly the same hash

23/54

Exploring History

`git log` - lists all commits made to a repository in reverse chronological order.

Flags	Function
<code>-p</code>	shows changes between commits
<code>-3</code>	last 3 commits, any number works
<code>--stat</code>	shows comparative number of insertions/deletions between commits
<code>--oneline</code>	just SHA-1 and commit messages
<code>--graph</code>	prettier output
<code>--pretty</code>	short/full/fuller/online

24/54

Exploring History continued

Flags	Function
<code>--since=X.minutes/hours/days/weeks/months/years</code> or <code>YY-MM-DD-HH:MM</code>	changes since
<code>--until=X.minutes/hours/days/weeks/months/years</code> or <code>YY-MM-DD-HH:MM</code>	changes before
<code>--author=<pattern></code>	contributor

25/54

Exactly what changes have you made?

- `git diff` – shows all changes that have been made from a previous commit, in all files
- `git diff R/modified.R` - see your changes of a particular file
- To see the differences between commits, use hashtags: `git diff 0da42ba 5m5lpac`
- The differences between commits for a specific file can be checked using `git diff HEAD~2 HEAD -- <file>`

26/54

Undoing/Unstaging

- There are a number of ways that you may accidentally stage a file that you don't want to commit

```
git add password.txt
```

- Check with status to see that it is added but not committed
- You can now unstage that file with:

```
git reset password.txt
```

- Check with status

27/54

Undoing: Discarding Changes

Perhaps you have made a number of changes that you realize are not going anywhere. Add what you ate for breakfast to `README.md`. Check with status to see that the file is changed and ready to be added

You can now return to previous committed version with:

```
git checkout -- README.md
```

Check with status and take a look at the file

You can return to a version of the file in a specific commit

```
git checkout m5ks3l8 README.md
```

If you want to correct the last commit message, do

```
git commit --amend -m "New commit message"
```

28/54

Undoing: removing from the repo

Sometimes you may want to remove a file from the repository after it has been committed. Create a file called `READYOU.md`, and add/commit it to the repository

You can now remove the file from the repository with:

```
git rm READYOU.md
```

List the directory to see that you have no file named `READYOU.md`. Use `git status` to determine if you need any additional steps

What if you delete a file in the shell without `git rm`? `rm README.md` What does `git status` say?

Oops! How can you recover this important file? `git checkout -- README.md`

29/54

Undoing: the big "undo"

It is possible that after many commits, you decide that you really want to "rollback" a set of commits and start over. It is easy to revert all your code to a previous version

You can use `git log` and `git diff` to explore your history and determine which version you are interested in. Choose a version and note the hash for that version

```
git revert b519sa4
```

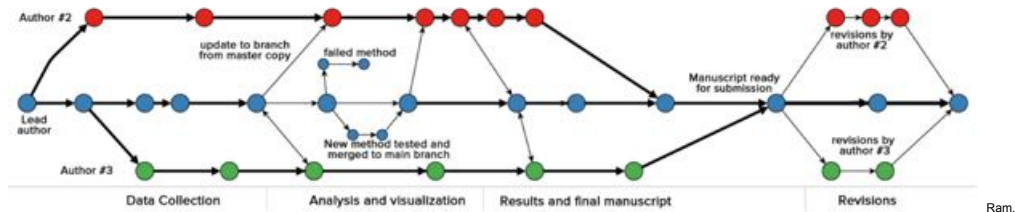
Importantly, this will not erase the intervening commits. This will create a new commit that is changed from the previous commit by a change that will recreate the desired version. This retains a complete provenance of your software, and be compared to the prohibition in removing pages from a lab notebook

30/54

Branches for Collaboration

Multiple authors can work on parallel branches, even on the same document

Conflicts must be resolved manually (using human intelligence)



Karthik. "Git Can Facilitate Greater Reproducibility and Increased Transparency in Science." Source Code for Biology and Medicine 8, no. 1 (2013): 7. doi:10.1186/1751-0473-8-7.

33/54

Resolving Conflicts

Conflicts may occur when two or more people change the same content in a file at the same time

```
Auto-merging README.md
```

```
CONFLICT (content): Merge conflict in README.md
```

```
Automatic merge failed; fix conflicts and then commit the result
```

34/54

Resolving Conflicts

The version control system does not allow people to blindly overwrite each other's changes. Instead, it highlights conflicts so that they can be resolved. If you try to push while there are some changes, your push will be rejected, need to pull first. Pull, conflicts, resolve manually.

```
<<<<<<< HEAD
```

```
Your current changes
```

```
=====
```

```
Conflicting changes need to be resolved
```

```
>>>>>>> dabb4c8c450e8475aee9b14b4383acc99f42af1d
```

GitHub

- GitHub is a hosting service where many people store their open (and closed) source code repositories. It provides tools for browsing, collaborating on and documenting code.
- Like facebook for programmers
- Free 2-year "micro" account for students - <https://education.github.com/>
- free private repositories. Alternatively - Bitbucket, GitLab, gitolite
- Exercise: Create a GitHub account, <https://github.com/>

37/54

Why use GitHub?

True open source

Graphical user interface for git

Exploring code and its history

Tracking issues

Facilitates: Learning from others, seeing what people are up to, and contributing to others' code

Lowers the barrier to collaboration: "There's a typo in your documentation."
vs. "Here's a correction for your documentation."

38/54

Remotes in GitHub

A local Git repository can be connected to one or more remote repositories

```
git remote add origin https://github.com/username/reponame
```

Check your work `git remote -v`

Use the `https://` protocol (not `git@github.com`) to connect to remote repositories until you have learned how to set up SSH

`git push origin master` - copies changes from a local repository to a remote repository

`git pull origin master` - copies changes from a remote repository to a local repository

39/54

Asynchronous Collaboration

- "FORK" someone's repository on GitHub – this is now YOUR copy
- `git clone` it on your computer
- Make changes, `git add`, `git commit`
- `git push` changes to your copy

*Create "NEW PULL REQUEST" on GitHub

40/54

Keeping in sync with the owner's repo

Add a connection to the original owner's repository `git remote add upstream https://github.com/username/reponame # upstream - alias to other repo`

`git remote -v` - check what you have

`git pull upstream master` - pull changes from the owner's repo

Make changes, `git add`, `git commit`, `git push`

Question: Where will they go? Can you do `git push upstream master`?

41/54

Create Pull Request

Go to your version of the repository on GitHub

Click the "NEW PULL REQUEST" button at the top

Note that the owner's repository will be on the left and your repository will be on the right

Click the "CREATE NEW PULL REQUEST" button. Give a succinct and informative title, in the comment field give a short explanation of the changes and click the green button "CREATE PULL REQUEST" again

42/54

What others see/do with pull requests

The owner goes to his version of the repository. Clicks on "PULL REQUESTS" at the top. A list of pull requests made to his repo comes up

Click on the particular request. The owner can see other's comments on the pull request, and can click to see the exact changes

If the owner wants someone to make further changes before merging, he add a comment

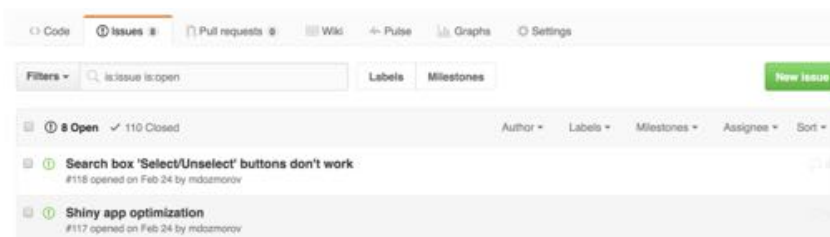
If the owner hates the idea, he just click the "Close" button

If the owner loves the idea, he clicks the "Merge pull request"

43/54

Track and Resolve Issues

- Issues keep track of tasks, enhancements, and bugs for your projects
- They can be shared and discussed with the rest of your team
- Written in Markdown, can refer to @collaborator, #other_issue

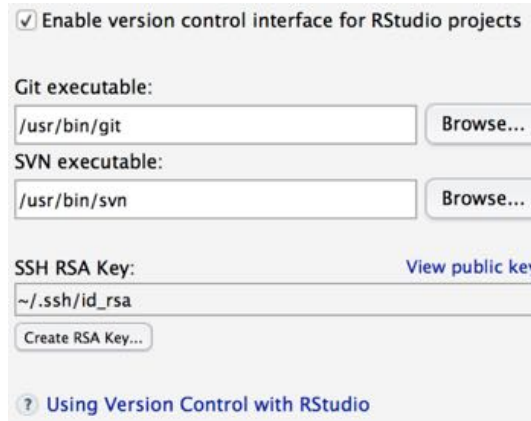


- Can use commit messages to fix issues, e.g., "Add missing tab, fix #100". Use any keyword, "Fixes", "Fixed", "Fix", "Closes", "Closed", or "Close"

44/54

RStudio and GitHub

RStudio has built-in facilities for git and GitHub. Set up git in Tools/Global Options



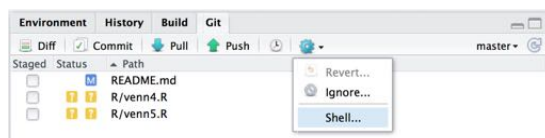
- Your Rstudio project can be your git repository
- Create a project with checkbox "Create a git repository" checked

45/54

RStudio and GitHub

Basic commands are available:

- See which files are untracked, modified, stage
- What branch you are in
- Add files, commit, push/pull
- See differences, history
- Revert changes, ignore files
- For heavy lifting git, use shell

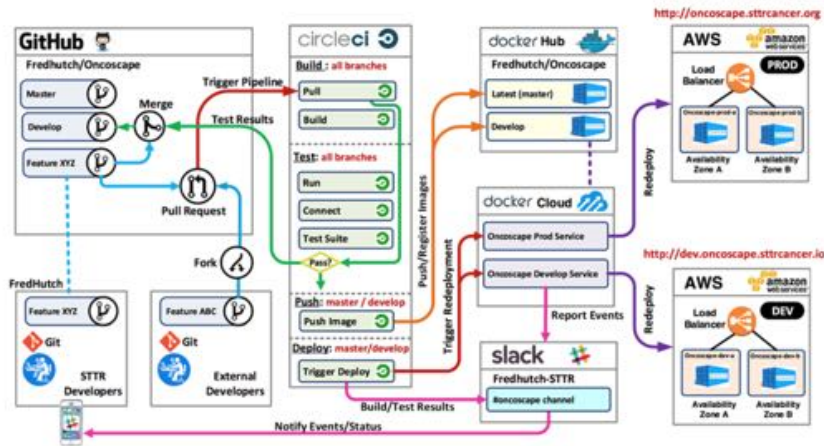


46/54

Big Data Analysis the Right Way

Oncoscape Integration and Deployment Pipeline

February 29th 2016



<https://aws.amazon.com/blogs/aws/building-bridges-for-better-cancer-treatment-with-the-fred-hutchinson-cancer-research-center/>

47/54

Encryption Concepts

- Both public and private keys are generated by one individual – they are yours
- A public key is a “lock” that can be opened with the corresponding private key
- Public key can be placed on any other computer you want to connect to, Private key stays private on any machine you’ll be connecting from
- Only your private key can “open” your public key



Private Key



Public Key

<https://wogan.me/tag/public-key-encryption/>

48/54

Getting Public and Private Keys

Generate your public and private keys

First, check if you already have them:

```
ls -al ~/.ssh
```

If not, generate:

```
ssh-keygen -t rsa -b 4096 -C your_email@example.com
```

49/54

Getting Public and Private Keys



```
~/.ssh/id_rsa    ~/.ssh/id_rsa.pub
```



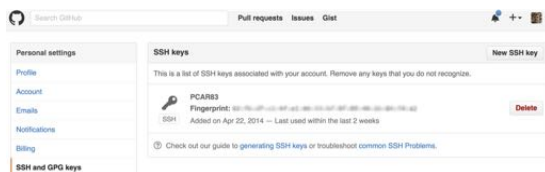
```
~/.ssh/authorized_keys
```

50/54

Add Public Key to GitHub

Go to your GitHub Account Settings

- Click “SSH and GPG keys” on the left, then “New SSH Key” on the right.
- Add a label (like “My laptop”) and paste your public key into the text box.
- Test it, `ssh -T git@github.com`. You should see something like “Hi username! You've successfully authenticated but Github does not provide shell access.”



<https://help.github.com/articles/generating-an-ssh-key/> <https://www.cs.utah.edu/~bigler/code/sshkeys.html>

51/54

Add Public Key to any Machine

- Copy your public key `~/.ssh/id_dsa.pub` to a remote machine
- Add the content of your public key to `~/.ssh/authorized_keys` on the remote machine
- Make sure the `~/.ssh/authorized_keys` has the right permissions (read+write for user, nothing for group and all)

```
cat ~/.ssh/id_dsa.pub | ssh user@remote.machine.com 'mkdir -p .ssh; cat >> .ssh/authorized_keys; chmod 600 authorized_keys'
```

52/54

Password-less login

Your private key should be visible to your terminal session

Start SSH agent

Add auto-start function in your `~/ .bashrc`

```
SSH_ENV=$HOME/.ssh/environment
function start_agent {
    echo "Initializing new SSH agent..."
    # spawn ssh-agent
    /usr/bin/ssh-agent | sed 's/^echo/#echo/' > "${SSH_ENV}"
    echo succeeded
    chmod 600 "${SSH_ENV}"
    . "${SSH_ENV}" > /dev/null
    /usr/bin/ssh-add
}

if [ -f "${SSH_ENV}" ]; then
    . "${SSH_ENV}" > /dev/null
    ps -ef | grep ${SSH_AGENT_PID} | grep ssh-agent$ > /dev/null || {
        start_agent;
    }
else
    start_agent;
fi
```

<http://mah.everybody.org/docs/ssh> <https://gist.github.com/reziam/850855>

53/54

Copy files over SSH

- `scp <file> user@remote.host.com:~/work/`
- `rsync -arvh --progress <file>
user@remote.host.com:~/work/`

54/54