Git it Together

Beginner Slides

Paul Johnson[1]

[1] Center for Research Methods and Data Analysis

2018

Outline

1. Motivation

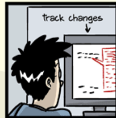
2. Git BASH: Gitting to Know You

3. 3 Common Scenarios
   - Scenario 1: Track one professor's GitHub repository
   - Scenario 2: Create your own Repository
   - Scenario 3: Interact with a Remote
   - Customs for Managing Branches

4. Conclusion

KU

Why Bother?



CRMDA has a Git page

https://crmda.ku.edu/git-help

- We have
  - a long-ish note/essay, "Git it Together! Version Management For Research Projects" CRMDA Guide #31, which also offers these slides
  - a short-ish note about using "GitLab: Instructions for Getting Started" CRMDA Guide #34
- And some scribbles (in the folder http://crmda.dept.ku.edu/guides/31.git) about details like
  - The "Large File Storage (LFS)" problem http://crmda.dept.ku.edu/guides/31.git/31.git-lfs.md
- There is an endless supply of Websites and self-help manuals for Git. A full-sized book named Pro Git (Chacon & Straub, 2014) is among the most helpful.

Cheat sheet

Basic Usage for Git version-tracking repositories

`git clone` - Copies a version-tracking repository (and all of its history). Usually for interacting with remote servers.

`git init` - Initiates a new version-tracking repository in current working directory.

`git pull/push` - Keep up to date with remote repository (retrieve/send).

`git add` - Tell Git to begin monitoring a file.

`git commit` - Tell Git to take a "snapshot" of altered files.

`git status` - Ask for report on files in project. Suggest "git status .".

`git log` - Ask for history report on project.

Git in a BASH Terminal Session

- Assume you have Git installed.
- If not, stop and install it. See instructions in chapter 3 of "Git it Together".
- Whether on Windows, Linux, or Mac, system will have a "Bash Terminal" where the user can interact with Git
  - Windows: In explorer, notice right-click "Git BASH Here"
  - On Linux or Macintosh, any Terminal will have access to Git if it is installed. Open terminal, run "`git --version`" to make sure.

Good Opportunity to Learn about the Terminal

- Biggest challenge for novices will be understanding "where am I" in the directories. Commands to run:

1. `pwd` Ask your system to print (`p`) name of the current working directory (`wd`)
2. `cd dirname` Change (`c`) current working directory (`d`) to `dirname`
3. `mkdir dirname` Creates a new directory named `dirname`

- For more command line advice, see
  - the "Intro Terminal" guides on my Computing-HOWTO pages
  - my "shell notes" from the Software Carpentry workshop: https://github.com/pauljohn32/sc_shell
- For now, just relax and believe this can work, *if you have some patience*.

Uncle Paul's Advice: Keep Your Git Together

- Figure out where you want to keep a folder named GIT
- Clone all Git repositories in there, *so you can always find them*
- My GIT repository is in my home folder at the top level, `/home/pauljohn/GIT`
- Perhaps a Windows or Mac user would like `/home/username/Documents/GIT`.
- **DO NOT**
  - put this in a Dropbox or other automagical network file server
  - allow any directories with spaces or non-alphanumeric characters.

Focus on 3 Common Use Cases

- In following sections, we consider 3 typical usage scenarios.
  1. Track a remote repository, keep up to date
  2. Create your *own* repository in your *own* computer
  3. Interact with a server by pulling and pushing changes
- We expect novices can follow along with the first 2 scenarios without too much trouble.
- The third, which puts the pieces together, will probably take some practice.

KU

Outline

1. Motivation
2. Git BASH: Gitting to Know You
3. 3 Common Scenarios
   - Scenario 1: Track one professor's GitHub repository
   - Scenario 2: Create your own Repository
   - Scenario 3: Interact with a Remote
   - Customs for Managing Branches
4. Conclusion

Retrieve a set of class notes

- Paul Johnson has a GitHub account named “pauljohn32”
- He keeps a couple of projects there, but not most things for CRMDA (for which we use a GitLab server that we administer)

Browse first

- Browse to `https://github.com/pauljohn32/RHS`

This repository Search | Pull requests | Issues | Marketplace | Explore

pauljohn32 / **RHS**

Unwatch 1 | Star 1 | Fork 1

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Exercises and Notes about Rabe-Hesketh & Skrondal Multilevel and Longitudinal Modeling Using Stata

Edit

Add topics

136 commits | **1** branch | **0** releases | **2** contributors

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

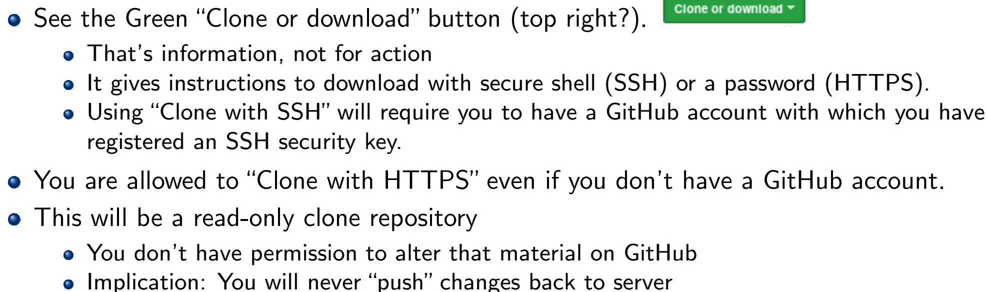
pauljohn32 downloader for smoking data — Latest commit df12636 4 days ago

exercises	downloader for smoking data	4 days ago
guides	Ex-05.1: some edits during class	11 months ago
notes	Chapter 13 notes	2 years ago
.gitignore	gitignore	a year ago
README.md	README update.	2 years ago

README.md

KU

Browse first

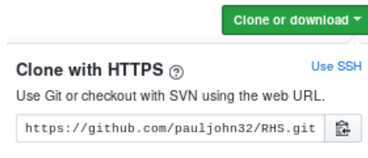


- See the Green "Clone or download" button (top right?).
  - That's information, not for action
  - It gives instructions to download with secure shell (SSH) or a password (HTTPS).
  - Using "Clone with SSH" will require you to have a GitHub account with which you have registered an SSH security key.
- You are allowed to "Clone with HTTPS" even if you don't have a GitHub account.
- This will be a read-only clone repository
  - You don't have permission to alter that material on GitHub
  - Implication: You will never "push" changes back to server

KU

Choose the HTTPS

- The small box under "Clone with HTTPS" will offer an address



  - That gives the address of the repository you want to clone

https://github.com/pauljohn32/RHS.git

Choose the HTTPS ...

- In your terminal, type this:

```
1  $ git clone https://github.com/pauljohn32/RHS.git
```

Here's the response:

```
Cloning into 'RHS'...
remote: Counting objects: 871, done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 871 (delta 21), reused 37 (delta 11), pack-reused 824
5  Receiving objects: 100% (871/871), 10.14 MiB | 7.81 MiB/s, done.
Resolving deltas: 100% (353/353), done.
```

KU

Inspect the result

```
$ ls
```

```
RHS
```

```
$ cd RHS
```

```
$ ls -lah
```

```
total 32K
drwxrwxr-x  6 pauljohn32 pauljohn32 4.0K Feb 17 12:18 .
drwxr-xr-x  3 pauljohn32 pauljohn32 4.0K Feb 17 12:18 ..
drwxrwxr-x 43 pauljohn32 pauljohn32 4.0K Feb 17 12:18 exercises
drwxrwxr-x  8 pauljohn32 pauljohn32 4.0K Feb 17 12:18 .git
-rw-rw-r--  1 pauljohn32 pauljohn32  242 Feb 17 12:18 .gitignore
drwxrwxr-x  6 pauljohn32 pauljohn32 4.0K Feb 17 12:18 guides
drwxrwxr-x  4 pauljohn32 pauljohn32 4.0K Feb 17 12:18 notes
-rw-rw-r--  1 pauljohn32 pauljohn32  810 Feb 17 12:18 README.md
```

Inspect the result ...

- The ".git" folder is the historical archive, it is your local "record keeper". **DO NOT DELETE IT!**

Use git log to review the history

- Default output is verbose, going from recent to past. To break off the page-by-page listing, hit the letter 'q' on the keyboard.

```
$ git log
```

```
commit df12636913e00881cb2b715339c1e41dd19b77b1 (HEAD -> master, origin/master,
    origin/HEAD)
Author: Paul E. Johnson <pauljohn@ku.edu>
Date:   Tue Feb 13 15:38:03 2018 -0600

    downloader for smoking data

commit 829a4d2bc298358b93a28c94d2f3471010ebb8af
Author: Paul E. Johnson <pauljohn@ku.edu>
Date:   Tue Feb 13 15:30:10 2018 -0600

    Ex-01.3

commit 99c14e866f4e12e597172948d184039c714c240e
Author: Paul E. Johnson <pauljohn@ku.edu>
Date:   Thu Feb 8 17:37:21 2018 -0600
```

Use git log to review the history ...

```
17      Ex-02.2-ghq:

        Ex-02.2-ghq: Rmd, R and html

        Ex-02.2-ghq: build out update

23  commit 65d8c484d0aab78eb3660dba7d142e3883026d7c
    Author: Paul E. Johnson <pauljohn@ku.edu>
    Date:   Thu Feb 8 10:22:01 2018 -0600

        Ex-02.1-pefr worked fully
```

- If you ask Mr Google or check stackoverflow.com for advice, you'll see that "git log" can be run with many arguments to beautify the output.
  - I generally ignore that.
  - I avoid customizing my user account with aliases and shortcuts.

Checking for Updates

- This Git repo is updated periodically, users will want the latest and greatest.
- Open a Terminal, and from the RHS directory, run:

```
$ git pull
```

- As long as you have not changed any files, or damaged settings in the .git directory, this will work
- Except if it does not (See next slide).

KU

There is only one thing to worry about: pull fails if you edit files

- The repository you have is a “read only” clone
- However, *You are Allowed to Edit* the files.
- **If**
  1. you edit a file, and
  2. the repo manager edits a file

  then “git pull” will fail.
- It fails because git wants to protect your edits, it does not want to erase them.

KU

How your edit causes conflict

- If you both edit "README.md", here's the error message that results

```
$ git pull
```

```
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 2), reused 3 (delta 2), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/pauljohn32/RHS
   df12636..37518f8  master     -> origin/master
Updating df12636..37518f8
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```

- Ignore the advice to "commit or stash", that is for contributors, not read-only repo watchers

How your edit causes conflict ...

If you want to edit a file

copy it to another file name and edit that

- If you forget that and do edit a file, then recover it.
  1. rename the file you edited
  2. recover the original copy from the history:

```
1 $ git checkout -- name-of-removed-file
```

  3. this will work, but not if you committed your changes.

KU

Outline

1. Motivation
2. Git BASH: Gitting to Know You
3. 3 Common Scenarios
   - Scenario 1: Track one professor's GitHub repository
   - Scenario 2: Create your own Repository
   - Scenario 3: Interact with a Remote
   - Customs for Managing Branches
4. Conclusion

Track your own effort, locally

- A remote server is not necessary
- This is just about keeping notes and history for yourself
- I tell students to track any project they start, no matter what.
    - If you don't track your effort, please stop and think if your effort is worthwhile.

KU

Step 1. Create a repo in a directory

- Start in your GIT directory
- Create an empty directory.
  - e.g.

```
$ mkdir fun1
```

  - change into that directory

```
$ cd fun1
```

- Run "git init" to create your local repo
  - If your directory is not in a network file server, run

```
$ git init
```

```
Initialized empty Git repository in /tmp/fun1/.git/
```

  - On a network file server, can allow teammates access by adding " –shared=group "

```
$ git init --shared=group
```

KU

Add some content

- Step 2. Add a file, any file you like
  - Use any editor you like
  - Windows may make this frustrating, try "`touch README.md`" first to create an empty file. Look at directory in file manager, then use Emacs, Notepad++, or some other adequate editor after that.
- Step 3. Tell git to track that file

```
$ git add README.md
```

- Step 4. Commit the file (means take a snapshot). **Don't run this yet!***

```
$ git commit README.md
```

- An editor will pop up. This editor, "vi", may be unfamiliar to you. If you want to avoid vi for now, add a commit message on the command line

```
$ git commit afile.txt -m "this is the new fabulous report"
```

Add some content ...

- Then edit README.md, then commit it again.
- Repeat several times

* **see next slide!**

If you run "git commit README.md" and vi opens . . .

1. hit the letter "i", which turns on insert mode. You should be able to type a message in first line of file. Only use keyboard arrows to move cursor. The mouse is not going to work.
2. After typing your message, hit these 4 keys (in sequence)
    1. "Esc" (The escape key on top left).
    2. ":" (the colon key: causes vi to be ready for commands)
    3. "w" (writes the file)
    4. "q" (quits vi)

My student jb sells needlepoint projects on Etsy:

Review History with "git log"

- The Git log

```
$ git log
```

```
commit 3e6a036db79705a5dcd0167bb312c98bb6a982f2 (HEAD -> master)
Author: Paul E. Johnson <pauljohn@ku.edu>
Date:   Sun Feb 18 12:29:04 2018 -0600

    Edit README a third time

commit 24b03668d86254ae2a44e47105bb3f047420ae4c
Author: Paul E. Johnson <pauljohn@ku.edu>
Date:   Sun Feb 18 12:28:36 2018 -0600

    edit readme

commit 620a52fbde42c366138e8709b54b08e7a5776c54
Author: Paul E. Johnson <pauljohn@ku.edu>
Date:   Sun Feb 18 12:28:12 2018 -0600

    readme added for git tracking
```

Review History with "git log" ...

- If you ask a randomly chosen Russian teenager, they will say run

```
1 $ git log --oneline
```

or

```
$ git log --oneline --decorate
```

KU

Status check

- This command surveys the project folder and makes a report (which will not be very interesting at the moment)

```
$ git status
```

```
On branch master
nothing to commit, working tree clean
```

- Make 2 changes
  - Add another file in the project. Any kind, any name (e.g., "iamasuperhero.txt")
  - Edit README.md and save it, but do not commit it.

KU

Status check ...

- After that, git status will be more interesting.

```
$ git status
```

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        iamasuperhero.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Cautions 1: don’t carelessly add material

- We do not add/commit
  - Trash folders, tmp files, backup files
  - password files, confidential client data
- To avoid accidental additions, **DO NOT add/Commit whole directories**. Add file-by-file.
- Difficult to completely expunge confidential information without ruining repository.

Cautions 2: Careful with "git commit -a"

- I suggest adding and committing individual files
- A shortcut to commit all revisions is to add "-a":

```
$ git commit -a
```

  - editor will warn you about changes, so not horribly unpredictable

- However, if you carelessly run this:

```
$ git commit -a -m "your message here"
```

  then you have no chance to review your actions.

- Suppose you have accidentally deleted or removed a file. Even if you don't explicitly run "`git rm filename`", Git will remove files from project when you run "`git commit -a`".

KU

Create a branch

- Suppose your repo files look good, but you want to work on a new feature

```
$ git branch pj-xfix
```

- Check your branch was created

```
$ git branch -avv
```

```
* master  3e6a036 Edit README a third time
  pj-xfix 3e6a036 Edit README a third time
```

- Change the working directory onto files tracked by the branch

```
$ git checkout pj-xfix
```

```
Switched to branch 'pj-xfix'
```

Create a branch ...

- Note that the untracked file, " iamasuperhero.txt " is still floating loose. If we edit it while in this branch, there will be trouble later.
- Do some edits. Add some files. commit the changes. Don't forget the commits, or else the edits are ignored.
- When it is perfect, put the revisions into the master branch

```
$ git checkout master
$ git merge pj-xfix
```

```
Updating 3e6a036..2d30518
Fast-forward
 README.md     | 2 ++
 newfile1.txt | 2 ++
 2 files changed, 4 insertions(+)
 create mode 100644 newfile1.txt
```

Create a branch ...

- Review the history:

```
$ git log
```

```
commit 2d3051817f91887d921613305943a32370f7bb2f (HEAD -> master, pj-xfix)
Author: Paul E. Johnson <pauljohn@ku.edu>
Date:   Sun Feb 18 12:37:33 2018 -0600

    README: edit inside branch pj-xfix
    newfile1: edit inside branch pj-xfix

commit 3e6a036db79705a5dcd0167bb312c98bb6a982f2
Author: Paul E. Johnson <pauljohn@ku.edu>
Date:   Sun Feb 18 12:29:04 2018 -0600

    Edit README a third time

commit 24b03668d86254ae2a44e47105bb3f047420ae4c
Author: Paul E. Johnson <pauljohn@ku.edu>
Date:   Sun Feb 18 12:28:36 2018 -0600

    edit readme
```

Create a branch ...

```
commit 620a52fbde42c366138e8709b54b08e7a5776c54
Author: Paul E. Johnson <pauljohn@ku.edu>
Date:   Sun Feb 18 12:28:12 2018 -0600
```

- Delete the (now unneeded) branch

```
$ git branch -d pj-xfix
```

```
Deleted branch pj-xfix (was 2d30518).
```

- Review

```
$ git status .
```

Create a branch ...

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        iamasuperhero.txt

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

- This was simple branch/merge because there were no other users involved.
  - We were sure master had not changed after we checked out the branch
  - There was no fear of conflicts between ourself on master and ourself on the branch

Future TODO for you: .gitignore

- As time goes by, your project folder may have files you don't want to track (such as backup files, error logs, photos of my lovely gardens, your fake ID from high school, or your most recent love letter to George Bush, etc.)
- Output from "git status" will always mention these files and suggest you track them.
- Create a file ".gitignore" in the top of the project to tell git to stop warning you about those files.

Future TODO for you: git rebase to squash commits

- You commit 10 times to correct 1 problem, 6 of which are goofups
- Clean up the project history by squashing those commits together

```
$ git rebase -i HEAD~10
```

  will launch an interactive session. Enter 1 commit ungoofed message.
- Read more: http://crmda.dept.ku.edu/guides/31.git/31.git-squash.md

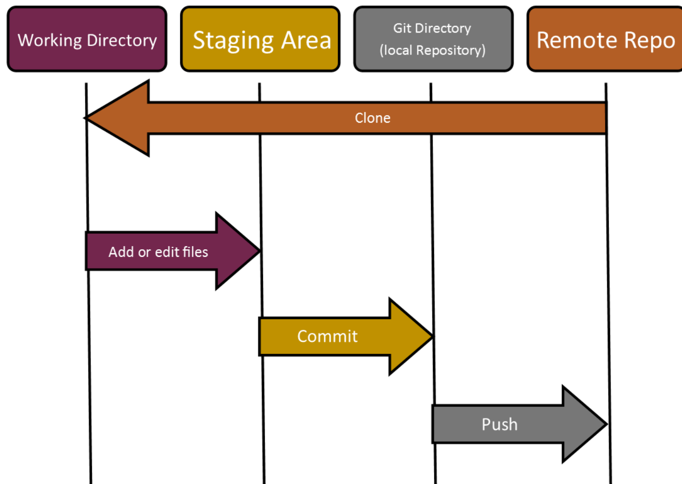
Outline

1. Motivation
2. Git BASH: Gitting to Know You
3. 3 Common Scenarios
   - Scenario 1: Track one professor's GitHub repository
   - Scenario 2: Create your own Repository
   - Scenario 3: Interact with a Remote
   - Customs for Managing Branches
4. Conclusion

Interacting with Teammates via A Remote Repo

- "git clone" retrieves a copy, as we have seen before.
- The git folder knows where its remote server is. Nickames it "origin"
- User has ability to "push" changes and "pull" updates.
  - After commits, "git push" sends changes to server
  - "git pull" fetches and merges updates from remote (some subtle problems come up, see "Git it Together").

Visualize your workflow



Remote magic, or not

- Setting up the remote repo requires some specialized knowledge.
  - A “bare” repository

    Definition: allows “clone”, “pull” and “push” commands.
  - Can be done, but not too easy to correctly regulate user access
- “GitHub”, “GitLab”, “BitBucket” simplify that with Web server GUI.
- Our workers in CRMDA almost NEVER need to create bare repos anymore.

Standard Workflow

1. Manager creates repo, tells teammates the address
2. Worker clones a copy of project

```
$ git clone <address provided>
```

3. Worker creates personal branch (with informative name)

```
$ git branch pj-docs
```

4. Inspect to see if branch was created

```
$ git branch -avv
```

5. Worker decides to work inside the new branch.

```
$ git checkout pj-docs
```

This turns the working directory into a view of files in the new branch.

Standard Workflow ...

Shortcut achieve both creation of branch and checkout in one step

```
$ git checkout -b pj-docs
```

6. User edits files. Uses "git add" or "git commit" as usual.
7. Send a copy back to the server:

```
$ git push -u origin pj-docs
```

8. Figure out way (or ask manager) to merge revisions onto the main project.

KU

Standard Workflow ...

Big Mystery

Question: How can the branch “pj-docs” stay in harmony with larger project?

Answer: this as much a social science as computer science problem! Must develop team expectations and cultivate communication

Review branches within local and remote repos

- Suppose there is no remote server. Then the branches are all local.

```
$ git branch -avv
```

```
* master                  819fb77 [origin/master: ahead 1] multilevel
      random-intercepts-2:
  pj-temp                 28f9b31 presentation/ordinalSEM.lyx: minor edits.
```

I've got the "master branch" (which is checked out) and a branch named "pj-temp" where I'm experimenting with a new feature

- Suppose now a remote is added and pulled. After that I see more branches

```
  * master             819fb77 [origin/master: ahead 1] multilevel random-intercepts-2:
  pj-temp                 28f9b31 presentation/ordinalSEM.lyx: minor edits.
  remotes/origin/HEAD     -> origin/master
4 remotes/origin/px-msha  c713097 msha/import: change outdir to workingdata
 remotes/origin/master    b806ee4 random-intercepts-1: 2018 style update
  remotes/origin/kk-maxilikeli d3ba2c4 hbsc-subset2-key2
 remotes/origin/red-gx    b30ccdc summeR-1.4 getwd insert initProjects
```

Review branches within local and remote repos ...

- “remotes/origin/xxx” are “remote branches”. I don’t edit them directly, they are copied from the server
- Suppose I run “ git checkout px-msha ”. After that, the output from “git branch -avv” will show a local “tracking” branch “px-msha”

```
 *px-msha         c713097 [origin/px-msha] msha/import: change outdir
 master         819fb77 [origin/master: ahead 1] multilevel random-intercepts-2:
 pj-temp          28f9b31 presentation/ordinalSEM.lyx: minor edits.
 remotes/origin/HEAD     -> origin/master
 remotes/origin/px-msha   c713097 msha/import: change outdir to workingdata
 remotes/origin/master    b806ee4 random-intercepts-1: 2018 style update
 remotes/origin/kk-maxilikeli d3ba2c4 hbsc-subset2-key2
 remotes/origin/red-gx      b30ccdc summeR-1.4 getwd insert initProjects
```

- The “local tracking branch” px-msha is currently synchronized with origin/px-msha. We can see that because the most recent commit is “C713097” for both.
- However, if I edit and commit in px-msha , it will be different than “origin/px-msha”

Review branches within local and remote repos ...

- And if the owner of px-msha makes changes from his computer and pushes to the server, then the copy on the server, which is referred to as "origin px-msha" is different from the others.
- Understand the effect of committing and pushing.
  1. commit: update information in `px-msha`
  2. push:
     1. updates `origin/px-msha` the local copy of the server branch
     2. copies branch to remote, thus synchronizing "origin px-msha".

fetch versus pull

- If we are in a branch, say master, and run "git pull", here is what git does
  - Retrieve the newest from "origin master"
  - save that in local "origin/master"
  - pull updates from "origin/master" and apply to master
- If we run "git fetch" the records on all of the remote branches are pulled, so "origin/master" (or any branch named "origin/xxx") is retrieved.
- fetch does not apply the changes in "origin/master" to "master" automatically.

KU

Branches Confusing, but Helpful

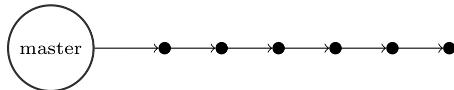
- We discuss recovering from mistakes in the manual *Git it Together*
- `origin/px-msha` is a "safe place" to revert to.
- If we fiddle with px-msha and make errors, it is very easy to reset `px-msha` to `origin/px-msha`.

Outline

1. Motivation
2. Git BASH: Gitting to Know You
3. 3 Common Scenarios
   - Scenario 1: Track one professor's GitHub repository
   - Scenario 2: Create your own Repository
   - Scenario 3: Interact with a Remote
   - Customs for Managing Branches
4. Conclusion

master and other branches

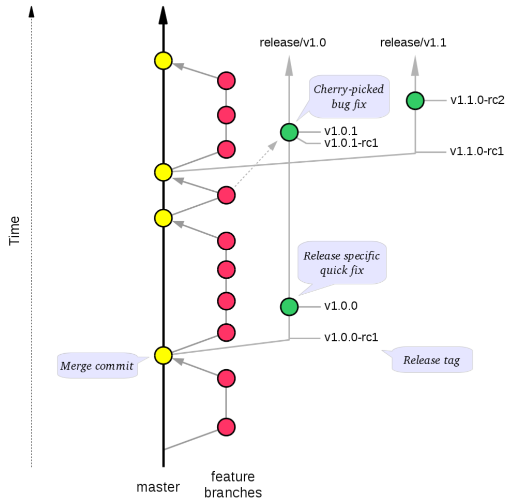
- All git repositories have at least one "branch", which is called the **master branch**



solid dots represent commits.

- For CRMDA, master is the "correct" "currently working" version of a project.
- We don't allow users to push onto the master branch.
- A project manager is supposed to make sure that master is good at any moment

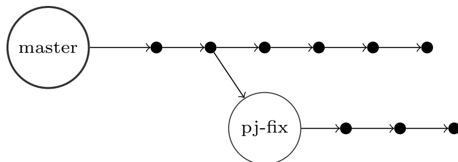
Visualize the branch & merge



The “stable mainline” branching model

Schematic overview

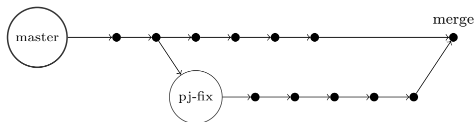
1. Users "git clone" our project repo
2. Create personal branch (`git branch pj-fix`)



3. Check out the branch (`git checkout pj-fix`)
4. Edit files, add, commit, etc
5. Push the branch onto the server (`git push -u origin pj-fix`) (see **caution** below)

Schematic overview ...

6. Request a merge onto the master branch



Caution: teamwork is difficult

- Workers create branches
  - `px-msha` / `red-gx` / `kk-maxilikeli`
- **Race to Push**: If one of the workers finishes her work first, and requests a merge onto master, and the manager does the merge, then

All of the other workers branches may become

1. *out of date*, **useless**, *irrelevant* or
2. *contradictory, harmful, conflicted.*

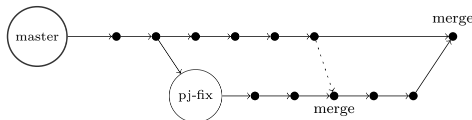
KU

How We Deal With That?

1. Avoid having different team members edit same files
2. Require team members to keep branches up-to-date with the master branch
   Use the magic 4 step sequence **OFTEN**:

```
$ git checkout master
$ git pull
$ git checkout pj-docs
$ git merge master
```

- The picture describing the branches will look like this



How We Deal With That? ...

- When merge request happens, the first thing the manager does is check to see if the branch is fully up to date with master branch. If not, reject.

KU

Preserve Sanity: short lived branches

- Name Branches in Obvious Ways (e.g. initials and purpose):

  pj-graphs

- Branches don't live forever. Merge and Remove branches!
- Users delete (prune) local copies of branches that were removed on server

```
$ git fetch -p
```

How does this look from a manager point of view?

- A worker says “I want you to merge px-msha”.
- The merge can be done in
  - GitLab graphical iterface
  - Command line on manager workstation
- On workstation, do this

```
$ git fetch
```

```
$ git branch -avv
```

- Inspect the changes

```
$ git checkout px-msha
```

- If satisfactory, merge `px-msha` and delete:

KU

How does this look from a manager point of view? ...

```
$ git checkout master
$ git merge px-msha
## remove old branch local and server
$ git branch -d px-msha
$ git push origin --delete px-msha
$ git fetch -p
```

- Danger Will Robinson: author of px-msha continues changing and pushing. My checked out local copy is stale.

```
$ git fetch
$ git checkout px-msha
$ git merge origin/px-msha
```

  - or, equivalently:

How does this look from a manager point of view? ...

```
$ git checkout px-msha
$ git pull
```

TODO for you: Git-LFS

- Git is very efficient with text files. It only stores changes between versions.
- Git is not efficient with binary files like movies, pictures, Excel sheets, etc.
  - Git will save historical copies of each version that is revised.
  - Storage will grow because .git folder keeps all old copies
  - Git will run slower and slower because of these hard-to-manage files
- Currently, solution we are using "Git-LFS", an add-on program that handles the binary files, keeps them in separate server.
- To read more: `http://crmda.dept.ku.edu/guides/31.git/31.git-lfs.md`

KU

Good chance to learn the "command line"

- All programmers & data scientists will need to go beneath the Graphical User Interface at times
- Using Git BASH in Windows offers a good way to get some practice because it is a general purpose shell.

KU

Practice is required

- Git will require practice, it is easy to forget “commit” “add” “push”.
- Users should create their own “cheat sheets” so they can remember what works.
  - Avoid blindly following advice you find in the Internet.
- For personal convenience on personal notes, I use SparkleShare (`www.sparkleshare.org`). It is an “auto git” program that always adds and commits all changes and copies them to the remote all the time.

KU

Git is a Priority for Career Preparation

- Git is an industry standard for data science researchers and programmers
- Most people who work in data science are expected to be familiar with Git, if not enthusiastic about it.
- Watch out for the koolaid, however. We've had several workers who become infatuated with Git jargon and the sprawling Internet community.

KU

References

Chacon, S. & Straub, B. (2014). *Pro Git*. New York, NY: Apress.