10/18

Ultra-Basic Guide to the Command Line and Git

Carl G. Stahmer

"Open Source Distributed Version Control"
-> Community developed piece of software that's free
-> Wide user database; it's pretty much top version control software
-> **Distributed** -> idea behind distributed system is that it doesn't just live on your own computer; Git can speak across in any direction (team collab projects- you can talk to a central server, you can talk to another person's computer, etc.-- it's very decentralized).
-> Very well designed to do some basic things very well but can build towards super advanced activities/goings on
-> **Version Control** -> Git is designed to help you get a handle on the fact that we all live in versions and we may want to save snapshots for some purpose and still track who changes what, when they change, how they change, etc.

Bitbucket, GitLab, Github -> hosting services/companies that provide a central place/central hub… common to use multiple remote services for different purposes

You have to download/install git onto their computer so you can use it

Steps:
Windows people will be asked about gitbash- yes you want it
Mac people -> might have to go into security preferences (privacy) and allow install
https://git-scm.com/downloads

Go to bitbucket.org and signup for an account

Get a coding environment/text editor - will really save you
Mac -> barebones bbedit (http://barebones.com/products/bbedit/)
Windows -> notepad-plus-plus.org

Command Line Essentials
File Window
Looking at folder/file organization
On Mac -> right click on the name of the folder at top of window and see it as a stack (this is the path)
Thinking through 'tree mode' is essential to coding/git
Windows -> c:// …

Root -> the beginning folder

Mac (Linux also works on Mac)

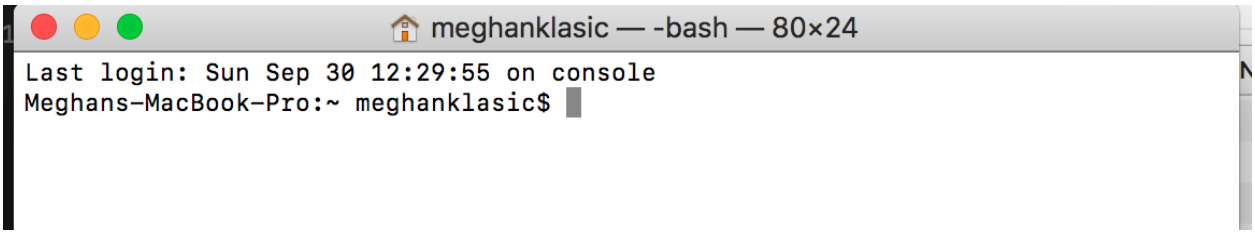
/Volumes/HD/Users/

Windows
C://Users/

Linux
/Users/

If you ever see:
~/files/
~ : is the home directory

Mac:
Applications/Utilities -> Terminal



Windows:
Programs -> GitBash

Mac:
pwd : present working directory (what you are currently in)
E.g. /Users/mghanklasic

ls: list -> gives list of all files in the folder
If you pull up finder window you see all the same things

ls -a: shows you all the invisible files in the folder
(a: all)

ls -l: shows you details on the files

Make a directory using command line (to put all git stuff):
mkdir git_workshop: make directory called git_workshop

cd: change directory
cd git (then hit tab and it will auto-populate)
or cd git_workshop/
-> now you're in this directory (if you do an ls it will be empty)

https://git-scm.com/book (everything we cover today is in this book- best beginner's guidebook to Git- goes beyond basic beginning; walks you through what most people will ever do in git) freely downloadable

Git: doesn't save whole versions of files, it saves a list of change instructions and stores them-- it means you don't end up with a ton of data on your drive.. Also helpful for collaborations - people to work simultaneously on the same file without over-writing each other
-> it's not google docs
-> you have to tell it i want to save this version (this is called a **commit**)
-> every time you **commit** you save something (it is not autosaved)

You have a **working directory (branch 1)**… git recognizes that things are changing but doesn't do anything with them... **branch 2 becomes the staging area**-- stuff you're telling git-- the next time I commit, these are things I want to commit (you don't have to put everything you change into staging)... **branch 3 is commit**
E.g. let's say you make changes to all three files in a directory but you only care about saving changes for two you can just move those 2 into staging

In Terminal:
mkdir temp : make a dummy temp folder
ls : shows you the temp directory
cd temp: go into temp folder

-> open text editor (bbedit)

Whenever we use git, we have to tell it we want it under version control -> commands begin with git

git init : git initialize
Meghans-MacBook-Pro:temp meghanklasic$ git init
Initialized empty Git repository in /Users/meghanklasic/git_workshop/temp/.git/
ls -a: will show you the hidden file directory
cd .git : will take you into the hidden file
ls : shows you what's in that hidden file

```
Meghans-MacBook-Pro:temp meghanklasic$ git init
Initialized empty Git repository in /Users/meghanklasic/git_workshop/temp/.git/
Meghans-MacBook-Pro:temp meghanklasic$ ls -a
.       ..      .git
Meghans-MacBook-Pro:temp meghanklasic$ cd .git
Meghans-MacBook-Pro:.git meghanklasic$ ls
HEAD            config          hooks           objects
branches        description     info            refs
Meghans-MacBook-Pro:.git meghanklasic$
```

Go back to temp directory

cd ~/git_workshop/temp

git status: best friend in the world- git will tell you about the current state of its environment
Right now, we have one branch (master) and no commits

Go to text editor
Type something and save to the temp folder

In Terminal:
-> if you type git status you'll see untracked files (in red)
-> if you then say **git add file1.txt** it will be
-> git status: see it turn green
-> git commit -m 'My first git training file' (adds a msg and commits)
-> git log: adds author, when commitment happened; see comment

```
Meghans-MacBook-Pro:temp meghanklasic$ git add file1.txt
Meghans-MacBook-Pro:temp meghanklasic$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   file1.txt

Meghans-MacBook-Pro:temp meghanklasic$ -m 'My first git training file'
-bash: -m: command not found
Meghans-MacBook-Pro:temp meghanklasic$ git commit -m 'My first git training file'
[master (root-commit) e689d1f] My first git training file
 Committer: Meghan Klasic <meghanklasic@Meghans-MacBook-Pro.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt
Meghans-MacBook-Pro:temp meghanklasic$ git log
commit e689d1fbb218b860b7233380f385cf69742aab24 (HEAD -> master)
Author: Meghan Klasic <meghanklasic@Meghans-MacBook-Pro.local>
Date:   Fri Oct 19 10:03:29 2018 -0700

    My first git training file
Meghans-MacBook-Pro:temp meghanklasic$
```

Now we want to change the file1.txt-- change in text editor and then save
Then go back to Terminal and put “git status” you will see the file says “modified: file1.txt” and it is in red meaning it’s been edited/changed

```
Meghans-MacBook-Pro:temp meghanklasic$ 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:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
Meghans-MacBook-Pro:temp meghanklasic$
```

Then you need to do git add to put into staging
Then you want to do git commit -m with a msg on the changes
Then use git status to see the commit codes for each change
Then use git diff and paste the commit codes so you can see the changes that you made in both of them

Repeat so you have 3 commits on the file

Rolling back to get to an earlier version
Head: top commit in your git log … it’s the top of the stack
Detached head:
I want to look at the file that was in my initial commit (let’s say 6 weeks have passed and i want to see what it looked like in my initial commit)
git checkout hash
Hash: code of the commit you want
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.
-> you’re working directory is no longer synced with this work (you’re in a whole other universe)
Good state to see what you had done initially

But now the file is changed in your working directory folder…

To get back:
git checkout master

do not commit in detached head state

Make new change to text file but it was a stupid change and I just want to get back to the head I had previously…

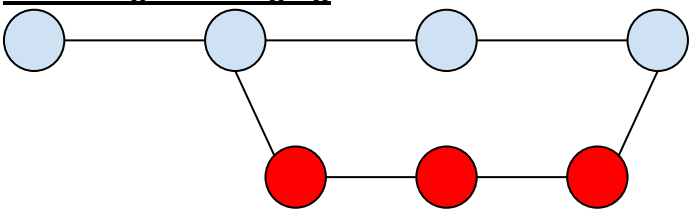
git reset --hard : takes all commits, staging, etc. to the last state

Side note:
I'm working on version 27 and i put in staging… but my commit is at version 26. But in staging I realize that I don't think what I did was waht I want to do but I do like the code so I want to save it somehow without losing commit 26

git reset --mixed: will result in V26 as my working directory file; V27 in staging; and V26 in commit… if you then do git commit you get V26 in wd, nothing in staging, V27 in commit but stacked below V26

git revert HEAD will always get you back to your last commit -> ensure you aren't in a detached head state (same as git reset --hard but reset gives you more control options)
**if you commit without a comment it will launch your text editor
-> use i (insert) and add your comment to the file
-> then use esc : w
-> esc : q
-> esc : q!

Branching and Merging



You can make a mirrored version of all the code at the point of a commit. From there you can flip back and forth between branches. If you work in the red branch it won't change/impact any of the master branch. Good for team work → I create code and then Andrew makes a branch so he can mess with the code and work on things…

Terminal:
git branch
-> you'll see * master (this all that's there)
To create a second branch with master:
git checkout - b *name* : creates a branch with whatever name you give it
git branch : shows master and dev now - the one that you are working within is green

Text Editor
Make some changes

Terminal

git status
git add
git commit

(your hard drive just sees one file- the change we just did)
But you can switch to your master branch
git checkout master
-> everything is as it was with the master branch
git checkout dev -> takes you back to the branch

Create new file in dev
Create new file and save
Do git add
git commit

Now- that file only appears as part of the dev branch
If you do git checkout master the file will no longer be there

Now- we want to bring branches back together and get into single branch
Move into the branch that you want to merge *into* (master branch here)
git merge *dev* (dev is the branch being merged in)

```
Meghans-MacBook-Pro:temp meghanklasic$ git merge dev
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.
Meghans-MacBook-Pro:temp meghanklasic$
```

File 2 is now in the folder but file 1 is in conflict-- so we have to deal with that
If you open File1 in your text editor you see:

```
Hello world I'm taking a git training.
Making some changes now to this file and then saving it

I'm trying some more stuff now.

<<<<<<< HEAD
I'm adding some stuff to my master branch now just to get crazy.
=======
Doing some more stuff to add on the dev branch.
>>>>>>> dev
```

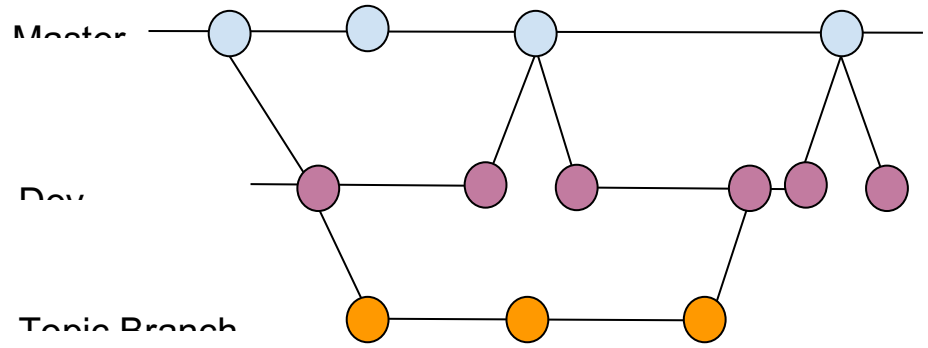
You could just delete the arrows and equals and save it but usually you will change some code or what not then save

Terminal
git add
git commit

Think of the master branch as the ‘release’ branch (when you are rolling it out or sending it out) - public stable version

The next branch that typically exists is ‘dev’ -> standard branch that everyone often works off of (good for team work)

Branches off dev are ‘topic branches’ -> any time someone is tasked with something… you might make a topic branch (for example calling it a task number branch)
-> work on topic branch until you’re ready-- then merge back into dev
-> maybe a lot of topic branch work happening and then they all get merged back into dev
-> then dev pushed back up to master



Now: How to work with central services (bitbucket; github; etc.)
Terminal
Get back to root directory
cd ~/git_workshop
ls : should be empty

Go to: https://github.com/cstahmer
Text mining repo: https://github.com/cstahmer/text_mining_with_r
We can use git to get all that data and take it
Go to clone/download and copy

Terminal
git clone *paste*
git clone https://github.com/cstahmer/text_mining_with_r.git

ls : shows whole repo