**Working with git**

**Set up git on your local machine**

1. **Mac and Unix**: get the git installer from http://goo.gl/xtfO and install git on your local machine. **Windows**: git for Windows is at git-scm.com/download/win; download the Git-xxx-yyy.exe installer file, and install it on your PC.

2. Make a folder to hold git repositories. The folder name and location don't actually matter, but it's useful if they are easy to find. On a Mac, a good choice is to open a terminal window and

cd ~

mkdir Repos

In Windows, create a folder called Repos on your C: drive, but NOT in a system folder (Users, Windows, etc.). The safest think is to put it at the top level, C:\Repos.

3. Open a terminal window\* on your local machine and use it to configure your username and email address. **These must match your Bitbucket account (login name and email) .**

(a) To configure your username, for example:

git config --global user.name "ellner"

(b) To configure your email address, for example:

git config --global user.email "spe2@cornell.edu"

(c) Configure your text editor. To choose (for example) emacs:

git config --global core.editor emacs

and similarly to chose Notepad or whatever. On a Mac you can use TextEdit:

git config --global core.editor "open –W –e"

If you do this, you will need to exit TextEdit with "command-S command-Q" (i.e.,

really quit TextEdit, not just close the document that is being edited).

\*In Windows, “open a terminal window” means: navigate to the main Repos folder or one of its sub-folders in Windows Explorer, right-click on the folder, and select ***Git Bash Here*** in the drop-down menu.

**Set up the repository on your local machine**

1. Open your web browser, and log in to your github account (www.github.com). Go to the Overview page for the bromecast-data repository, find the box at top-right that says “CODE”; click on that and then highlight and **copy** the git path (but don’t paste it anywhere yet). It looks like:

https://github.com/pbadler/bromecast-data.git

2. On your local machine, open a terminal window and go to the repository directory:

cd Repos (or maybe cd repos, depending on your initial setup)

3. Paste the path line into the terminal. In front of it git clone. So you create your local repository with a command like

git clone <https://github.com/pbadler/bromecast-data.git>

This will create a local copy of the repos. You should now see a new directory called bromecast-data which is your local working Git repository.

**Interacting with your local repository**

The first step is always to open a terminal window and cd to your Repos/drivers directory (in Windows: navigate to Repos/drivers, and open a Git Bash window).

1. Before you start work, make sure to get a fully up-to-date copy of the Github repository, which includes all the changes that others have made, so your changes will be added to theirs. The command to do this is

git pull

If there’s nothing new on Github, git will tell you that. You can then work with files in the repos using your usual tools (R, text editor, TeX, Word, etc.) in the usual way.

2. To see the status of your local repository, open a terminal, cd to the repository folder, and do

git status

This shows you what files git is monitoring. Ones you have changed recently will be flagged as modified.

If git status tells you that a file is Untracked (which is often the case when you create a new file), this means that git will ignore it: it won’t get copied to the repository, on your machine or on Github. To tell git that you want a file tracked, you need to add the file. For example, if you have just created chapter3.tex and you want it tracked, you add it by doing

git add chapter3.tex

If you want to add all your new files you can do git add –a or git add –all. To be more specific about what you’re adding, you can do things like

git add \*.tex

git add figures/\*.pdf

3. To preserve changes that you make to a file, you need to **commit** it to the repository. A commit requires a “message” describing the changes you’ve made. So for example to commit your changes to chapter3.tex you can do

git commit chapter3.tex –m “update to chapter 3”

If you leave out the –m “message” part, git will have your chosen text editor open a new file, into which you write your message and then close the file. In this case, the command-line version is easier.

To commit all the changes that you’ve made since the last commit (which is usually the case) do

git commit –a –m “your message”

4. You can go through several add-commit cycles as you work on your local copies of files. Eventually you need to share this work, by **push**ing the changes up to Github. Open a terminal, cd to the drivers repos folder, and do

git push origin master

All changes in your local repository will now be reflected in the repository on Bitbucket.

**Recovering from mistakes**

If you realize that the changes you’ve made since the last commit are a mistake, and you would like to make them go away, the command to do that is (be sure to include the period)

git checkout -- .

This only affects changes in the current directory, and its sub-folders. To make this act on the whole repos, change to the root folder of the repos.

Worse yet: suppose that you commit some changes, push them to Github, and **then** you realize that you made a horrible error like deleting a block of text by mistake. Or even worse, you discover that this happened a week (and several pushes) ago.

A solution is provided by the git **show** command, which lets you see or retrieve files from any branch, commit or tag on the online repository. You can see the status of files in the selected branch, commit or tag without moving them out into your working tree, or retrieve a copy of the old file *under a new name*.

# [reference] can be a branch, tag, HEAD or commit ID

# [filename] is the filename including path

# To just take a look

git show [reference]:[filename]

# To make a copy to copiedfile.txt

git show [reference]:[filename] > copiedfile.txt

**Example:** git show ec12d03:c8/chapter8.tex > oldch8.tex

The reference for a commit (like ec12d03) can be found by browsing to the main page for the repos on Github. This means that any past version of any file, even one that was deleted, can be recovered at any time.

**Resolving conflicts**

A conflict can arise if two people are simultaneously editing the same lines in a file. After the other person pushes their changes, you cannot just pull from the repo: when this is tried, you get a CONFLICT message about those files, and your local copies of the conflicted files will have both versions of the changes, indicating which branch the two versions come from.

One option is to squirrel away your files some place where git is not watching (so you have a copy with your changes) and then over-write your copy of the repos with the version on bitbucket. This can be done with

git fetch origin

git reset –-hard origin/master

(instead of master, it can be the name of whatever branch you want to do this in). You can then re-do your changes, starting from the updated local copy, commit them and push them.

I don’t yet know of another option.