git and rmarkdown cheatsheet

1 GitHub

GitHub is a widely used platform for open sharing of data, software, analysis code, and other materials. It uses software called **git**, which a powerful tool for organizing, collaborating on, and version controlling a project. We'll use GitHub to host our example project from OSF and then use git on our own machines to showcase some of its features.

1. Create a GitHub account at `https://github.com`
2. Once setup, push the green `New repository` button
3. Given the repository a name. It can be anything. "nottingham", "first-project", etc.
4. You can give it a description if you want.
5. Click `Create repository`. You're all setup!

2 GitKraken

git is the software that powers GitHub and which we will use to *version control* our project content, experiment with that content, and possibly introduce collaboration. There are lots of different tools for working with git, the simplest of which is a command-line interface that can be used via Terminal, Command Prompt, bash, etc.[1] For today's workshop, however, we will use a (free) product called GitKraken that provides a point-and-click interface for git. Set it up as follows:

1. Download GitKraken from `https://www.gitkraken.com/` and complete the installation
2. Open GitKraken and log in using your GitHub login details
3. Click `File > Init Repo`
4. Enter the folder containing the example project as the `New repository path`
5. Click `Create Repository`. You should see all of the project files in green near the centre of the screen.
6. On the left side, click the `+` symbol next to `Remote`
7. On the pop-up window, click `URL`.
8. For `Name`, type "GitHub". For `Pull URL`, type "https://github.com/`user/project`"

Now we have a Github "remote" repository setup and the local git repository setup with GitKraken. This means we're ready to start working with our project.

[1]You can find installation instructions for it here: `https://git-scm.com/downloads`

3 git Basics

Let's put git to work for us identifying when files have changed, recording those changes as *snapshots*, and navigating between versions. To that, we need to know some actions that git can perform. The key idea is that we will create snapshots of our project by **committing** files to the git repository and those changes will be reflected on Github (and to any collaborators) when we **push** them. Here's a glossary, which we'll walk through together:

- **stage**: select files to be recorded in a "snapshot" of the project
  - **unstage**: remove files from the snapshot (but not from your computer)
- **commit**: record a permanent snapshot of the staged files, labelled with a "commit message"
  - **amend**: modify (typically the most recent) commit with new changes or commit message
- **branch**: produce a complete local copy of the project where changes can be made independently of the "master" branch
- **merge**: update a branch with changes from another local branch (or a collaborator's changes from GitHub); you can change multiple branches independently.
- **push**: send the project (any new commits) to a remote server (like GitHub)
- **pull**: grab new commits from a remote server

GitKraken provides a point-and-click interface for all of these steps, but you can also execute them via the CLI commands `git add` (stage) or `git rm` (unstage), `git commit`, `git branch`, `git merge`, `git push`, `git pull`, etc.

Test Yourself

You get a hang of what git can offer by trying out the following.

1. Stage and commit your files
2. Change one or more files in any way. See how those changes become "unstaged".
3. Create a new branch to contain these changes.
4. Stage and commit the changes in the new branch.
5. Push the new branch to GitHub. See that they show up there.
6. Merge the changes to the master branch locally.
7. Push the changes to the master branch on GitHub.
8. Change some files through the GitHub web interface. Pull them locally.

Key Steps

Overall, the most essential steps of git are `pull > stage > commit > push`.
As a mnemonic device, maybe try "**P**eter **S**hould **C**ome **P**romptly".

4 Reproducible documents with rmarkdown

Once you've mastered git for managing and tracking your files, another useful set of tools to know relate to reproducible, dynamic documents. We'll use Rmarkdown as an example for this.

~~~
---
title: "An Rmarkdown Example"
author: "Thomas J. Leeper"
date: 2018-01-30
output: pdf_document
---

```{r data, echo=FALSE, results="hide"}
library("knitr")
opts_chunk$set(echo=FALSE)
library("gapminder")
library("ggplot2")
suppressPackageStartupMessages(library("stargazer"))
```

There are some descriptive statistics about the data:

```{r table1}
knitr::kable(aggregate(cbind(lifeExp, pop, gdpPercap) ~ year, data = gapminder, FUN = mean))
```

Figures 1 shows some general patterns.

```{r figure1}
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp, colour = continent)) +
  geom_point()
```

And the main regression are shown in the table.

```{r table2, results = "asis"}
stargazer::stargazer(
  x1 <- lm(lifeExp ~ I(pop/1000000) + gdpPercap, data = gapminder),
  x2 <- lm(lifeExp ~ I(pop/1000000) + gdpPercap + year + factor(continent), data = gapminder),
  header = FALSE
)
```

```{r inlinenumbers, results="hide"}
pop_effect1 <- sprintf("%0.2f", coef(x1)[2L])
pop_effect2 <- sprintf("%0.2f", coef(x2)[2L])
```

To summarize, here are some main results. To summarize, here are some main results.
The estimated effect of population size on life expectancy is `r pop_effect1`.
Using continent and year controls, however, changes this effect to: `r pop_effect2`.
~~~