git and rmarkdown cheatsheet

1 GitHub

GitHub is a widely used platform for open sharing of data, software, code, and other materials. It uses **git**, a powerful tool for organizing, collaborating on, and version controlling a project. Let's get started:

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. "amsterdam", "rt2-example", etc.
4. You can give it a description if you want.
5. Click `Create repository`. You're all setup!

2 Basics of using git (on the command line)

Now we'll work *locally* with git. To get stared, 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 local git repository and those changes will be reflected on Github when we **push** them. Here's a quick glossary:

- **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 snapshot of the staged files, labelled with a "commit message"
  - **amend**: modify a 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

You can implement these steps through RStudio, through Git GUI, GitKraken (see below) or the command line. On the command line, the key things to remember are: `git add` (stage) or `git rm` (unstage), `git commit`, `git push`, `git pull`, `git branch`, `git merge`, 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.

Overall, the most essential steps of git are `pull > stage > commit > push`.

3 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: 2018-04-06
output: pdf_document
---

```{r data, echo=FALSE, results="hide"}
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. And the main regression are shown in the table.

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

```{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`.
````