Useful `bash` commands

Handle installed packages during a major Debian/Ubuntu release upgrade

The situation

You want to upgrade your Debian/Ubuntu system to a major release upgrade. Such a release update includes that all data on your current disk will be overwritten. You need to back up your system, and want your new system to work just like the old system. So you have many packgages installed on the old system (manually, incl. PPAs, dpkg, apt-get, etc.), and now want the install those same packages on the new sytem.

Ways to achieve this

A pedestrian way to achieve this is to compile a list of all (manually) installed packages in a text file, and prepend each line with the installation command. Here are three commands that collect packages and could be used to in a bash script to re-install those packages. At the same time they illustrate the problem: they all return a different number of packages.

```
apt list --installed | wc -l
apt list --manual-installed | wc -l
aptitude search '~i !~M' -F '%p' --disable-columns | sort -u | wc -l
```

So what command returns the *right* packages? There are multiple ways to achieve this. I chose one of the suggested procedures outline in that link:

1. Use `apt-clone`. Install it by

   ```
   sudo apt-get install apt-clone
   ```

2. Create a backup with

   ```
   sudo apt-clone clone path-to/apt-clone-state-ubuntu-$(lsb_release -sr)-$(date +%F).tar.
   ```

This dynamically forms the release number in short form and appends it the name of the backup archive file (using the `lsb_release` command).

3. Restore backup

   ```
   sudo apt-clone restore path-to/apt-clone-state-ubuntu.tar.gz
   ```

4. Restore to newer release

```
sudo apt-clone restore-new-distro path-to/apt-clone-state-ubuntu.tar.gz $(lsb_release -sc)
```

Manually backup your data (WIP)

The home directory contains a lot of settings, data, perhaps executables, code, etc. WIP

All things `grep`...

Searching for strings in all or specific files

Search recursively through all files that end on `py` or `sh`, * `-R` (includes symbolic links) or `-r`: recursive * `-n`: line number * `-w`: match whole word * `-l`: (lower case L) can be added to provide file name of matching file

```
grep --include=\*.{py,sh} -rnw . -e '.boto.'
```

Git and GitHub

Create a new branch on local `my-name/my-feature`.

```
git branch feature/mjkrause
git checkout feature/mjkrause
```

You create a branch and committed to remote on one machine, now you want to work on it on a different machine

Suppose you create a branch in a repo on your local machine at work. At home you want to keep working in that branch, you clone the repo. And now comes the significant bit: don't just `git checkout _branch-name_` as that would result in a detached HEAD state and you wouldn't be able to commit to that branch. Instead execute

```
git branch --track branch-name origin/branch-name
```

That should return a message saying that you now track that branch from origin. Follow up with a `git branch` and you should see *branch-name* listed. If you now

```
git checkout branch-name
```

it should return a message saying that you switched to that branch and that it is up to date. You can start coding and committing to that branch now.

You wish to know the Git root directory name (the one which contains the file `.git`).

This likely is the name of the repository it was cloned from.

```
basename `git rev-parse --show-toplevel`
```

A quick way to this is by

```
git checkout -b feature/mjkrause
```

Rebasing

The situation (1), the problem (2, 3) and the solution (4)

1. Originally, you branched off of develop:

```
a -- b                          develop
   \
    f -- g                  your_branch
```

2. Then someone else branched off of develop, and merged branch1 back into develop before you did. Now develop is in a different state compared to you first branched off of it.

```
a -- b -- c        d            develop
  \    \      /
   \    d -- e              branch1
    \
     f -- g                 your_branch
```

3. To have the features containde in branch1 into your feature branch you need to rebase on develop.

```
a -- b -- c        d            develop
       \      /
        d -- e              branch1
```

4. The solution is rebasing your changes onto the new state of develop. This is done by taking off your changes, fetching the latest state of develop, and then replaying your changes back onto that new state of develop - done.

```
a -- b -- c        d            develop
       \      / \
        d -- e   \          branch1
                  \
                   f -- g   your_branch
```

First, checkout out develop and make sure that your version (local) is the same as the remote (origin) version. Run `git status` and look for the message *your_branch* is up-to-date with *origin/your_branch*. If not, make it so by

```
git fetch origin
git reset --hard origin/develop
```

Note that this is going delete all changes you may have (accidentally?) added to branch `develop`. But if that's not a problem you can just go ahead. Now change back to your branch and start the rebase process by

```
git rebase develop
```

This hopefully will have no conflicts. If so, run again `git status`. That likely returns *Your your_branch and 'origin/your_branch' have diverged, and have* x *and* y *different commits each, respectively.* That's a problem. Now you need to re-write history by forcing a push.

```
git push --force-with-lease
```

and run `git status` again. Confirm that the local and remote version of that branch do not diverge anymore. Your done.

If you feel adventurous...

You can run instead of that last command

```
git rebase -Xtheirs develop
```

and your feature branch will now contain those changes that have made on the original branch after you branched off of it to create your feature branch. This was introduced with git version 1.7.3. That replays again your changes onto the current state of `develop` and seems to skip a few steps. I would recommend trying this first on a new feature branch and convince myself that it works before trying it on *your_branch*.

Changing the base of your feature branch

Suppose `feature/my_feature` was based on `oldBase`, but now you want it to be based on `newBase` (e.g., `develop`). You can change the base of a feature branch by

```
git rebase --onto newBase oldBase feature/branch
```

[TODO: In GitHub `upstream` usually is `develop`. Suppose you need to keep up to date with changes in a different branch, say `feature/commons`. In that case set your branch to that:

```
git branch --set-upstream-to origin/feature/commons feature/new_feature
```

It will print `Branch feature/new_feature set up to track remote branch feature/commons from origin.`]

After that execute

```
git pull
```

Print history of commits, with the most recent one on top of the screen:

```
git log
```

If you use it with the `--graph --decorate` options the output includes the branch tree in color.

List all branches and who last committed to that branch

```
git for-each-ref --format='%(committerdate) %09 %(authorname) %09 %(refname)' | sort -k5n -k
```

We can't infer who is the author of a git branch. Git branches are pointers to a commit. As such it is not possible (not to my knowledge) who created a branch.

Show the the name or organization a repo was originally cloned from

```
xdg-open `git config --get remote.origin.url`
```

This opens the organization with the repository in a new browser window.

Deleting a branch

To delete a branch push it to remote first, then delete on remote, and last delete it on local. Suppose the branch we want to delete is `feature/my-branch`. Most times the name of the remote is `origin`, but it doesn't need to be. Option `-D` is forced delete or `--delete --force`.

```
git push --delete origin feature/my-branch
git branch --delete feature/my-branch
git branch -D feature/my-branch
```

Following this execute

```
git fetch --prune origin
```

Updating remote branches on local

Once you have merged a branch on the webinterface and you deleted the feature branch right there, you will still see that feature branch after `git branch -r` on your local. To refresh the tree on your local issue

```
git -c core.quotepath=false fetch origin --progress --prune
```

If you check again for remote branches you should not see the current status of feature branches. You still need to delete that branch locally as described in the previous paragraph.

Squashing commit messages when `Squash and merge` isn't made available in repo

This describes the action of squashing multiple commmits into a single commit. First get an estimate on how many commits are affected. In the GitHub UI that number shows up in the tab. Look at the commit log to figure out the first (i.e., oldest, usually the first commit in that branch after you branched off of, for instance, `develop`) commit that should be included in the squashing action. Strategies to find that first commit include reading the log (`git log`, perhaps with the `--pretty=oneline` option) and identify the number. Once found, verify that number. Suppose it's 64. In that case do

```
git rebase -i HEAD~64
```

where `-i` stands for *interactive*. That command opens the editor (e.g., emacs) that contains your last 64 commits to that branch, in the time-ascending order (so the last commit is at the end of that list). Each line in that document is prefixed with `pick`. Find the first commit where you want to start squashing (should be the first line if counted correctly). Leave this line alone, and start replacing `pick` with `s` in each other line to the last line of the commits.

Save the file and close it. This will open a second editor window with all commit messages and some other information. Delete the entire content and replace by the commit message for the squashed commit. Save and close.

Now the local copy of the branch will differ from the remote copy. Therefore you need to force the push like so

```
git push origin feature/my-branch --force
```

`push` copies your local branch into the remote repository. It is good practice to create a test branch, and perform the above steps on it. Suppose that branch is called `delme`. Once done with the process on `delme` do

```
git diff feature/my-branch my-original-branch
```

to see the difference between the two branches. Only make sure that before you start working on the actual branch to reset to the state of the remote, and to delete the test branch.

How to save the day when things went wrong...

You can save the day in multiple stages. After issueing the `git rebase -i HEAD~<some number>` and you think you made a mistake you can delete the content of the file, save it, and close the editor. That should output `Nothing to do` in the terminal.

If the original state of the branch (i.e., the state before you started the rebase) you are working on has been pushed to the remote already, but hasn't been

pushed following the rebase you restore the original state in the local branch using one of two strategies:

1. you reset the local branch to the state of the remote branch by

   ```
   git reset --hard origin/mkrause/branchname
   ```

2. you delete the local branch and checkout the remote branch

   ```
   git branch -D branchname
   git checkout remote/branchname
   git branch branchname
   ```

Using `git apply`

This command can take a patch file, which contains changes in a feature branch in mailbox format, and apply it to the current branch. For instance, suppose you're in branch `my_botched_branch`. For some reason you can't merge it easily to a target branch. Suppose you have branched off `develop`. In that case you could pipe the changes in the files of `my_botched_branch` to a file

```
git diff develop my_botched_branch > my_patch.patch
```

Now checkout `develop` and create a new feature branch `my_branch`, checkout this branch, and apply the changes in `my_patch.patch`

```
git apply my_patch.patch
```

This method can be a good choice if you find yourself in some mess with rebase or merge with an existing branch.

Showing only the headlines of commit messages

```
git log <some-old-commit>..<latest-commit> --pretty=oneline
```

The commits can also be tags or release numbers.

Getting a specific file out of an archive

Suppose you have an archive in the directory `myarchive.gz`. The first three lines list all files in that archive. `myarchive.gz` could be buried deep in a file structure, for instance `root/files/foo/bar/`. You would learn about that path by listing the file. The is important as the entire path is the second argument for the specific file you wish to extract. The last line would extract `myarchivefile.gz` from the original archive into the present directory.

```
tar -tf /foo/bar/baz/myarchive.gz
tar -tf /foo/bar/baz/myarchive.gz | grep 'arch.*gz'
```

```
tar -tf /foo/bar/baz/myarchive.gz root/files/foo/bar/myarchive.gz
tar -xf /foo/bar/baz/myarchive.gz root/files/foo/bar/myarchive.gz .
```

Copying a file from a VM to another server using `rsync`

Now suppose you want to copy the file you extracted above to the local server

```
rsync -avz username@server:/home/username/foo/bar/baz/root/files/foo/bar/myarchive.gz .
```

So again, you need to use the complete path information of the file (i.e., the location to which you wrote it to in the above step).

`cron` and `crontab`

The UNIX / Linux scheduler can be tricky. For instance, a script that runs just fine in a terminal may not run in the same way in `crontab`. Some things need to be taken care of beforehand.

Suppose you want to schedule running a shell script, and suppose your script relies on a bunch of environment variables, and perhaps some other utilities, such `jq`. This requires some extra work.

Start out by creating a simple schedule that runs every minute, such as

```
* * * * * env > /tmp/my.env
```

That writes the environment variables into the file in the temporary directory. Run `tail -f /tmp/my.env`: you see that there's isn't much. So if you need some environment variables you best have them saved in a file (e.g., `.env`) in the project root or somewhere and source them.

The cron shell isn't an interactive shell or a login shell. Neither does it run `bash`. So to make bash commands have effects, paste the following on top of the cron file. Open the cron file with `$ crontab -e` (should open in your favorite editor). The paste something like

```
SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/path/to/your/script:/snap
SCRIPT_DIR=/path/to/your/script/
```

The first line makes bash the default shell to execute commands in that file. To find out what `PATH` should contain, run `echo $PATH` in a terminal and copy the output from it to paste it into the crontab file. If cron needs to find executables, say `jq`, run `$ which jq`, and add the output to `PATH`.

The shell cron uses is the Bourne shell, and it uses `sh` to execute commands, but I think it's easier to just use bash instead.

In the actual cron scheduling command

```
# Run every day at midnight.
0 0 * * * . $SCRIPT_DIR/.env && cd $SCRIPT_DIR && my_awesome_script.sh $ARG1 $ARG2 $ARG3 > /
```

Perhaps you run that script using `.` or `source` in a terminal. Here in `crontab` that's not necessary. `$ARG1 $ARG2 $ARG3` are assumed to have been loaded by loading `.env`. Redirect the output to standard error, unless you want cron to email it to somewhere (which requires setting up some mail service). When done, save the file and close the editor. The cron job should run immediately.

Logging cron jobs

First, convince yourself that `cron` is running at all by

```
sudo systemctl status cron
```

or `sudo service cron status`. If it is running there are several ways to monitor what's happening in the cron job. One way is

```
sudo grep -i cron /var/log/syslog`
```

is one. But it is best to enable output to `cron.log` to monitor only cron-related messages. To this open

```
sudo vi /etc/rsyslog.d/50-default.conf
```

and uncomment the line

```
#cron.*   /var/log/cron.log
```

and monitor it by `sudo tail -f /var/log/cron.log`.

Using cron jobs in Docker containers

Using one container to run a cron job is probably the best. Running more than one job in a Docker container is an anti-pattern, and doesn't follow the best practice of running on job per container. But sometimes it is the easiest.

How to configure the crontab file in a container depends on the base image. For instance, if the base image is slim, stretch, alpine, etc, so not a full-blown OS, `cron` isn't even installed. So you need to install it with

```
.
.
.
RUN apt-get update -qq \
  && apt-get install -qq \
    cron rsyslog \
  && apt-get clean \
  && rm -rf /var/lib/apt/lists/*
.
```

.
.

and probably a bunch of other packages as well. Have the crontab file (e.g. `docker_crontab_file`) reside in the same directory as the Dockerfile. Then

```
ADD docker_crontab_file /etc/cron.d/some_cron
RUN chmod 0600 /etc/cron.d/some_cron
RUN crontab -u root /etc/cron.d/some_cron
```

The first command adds that file to a directory (which it creates) in the container. The second line sets permissions to `rw` for the owner. The file permissions for the crontab file could be critical.

Most importantly in the third line we specify that `root` will be using that crontab file we add to `cron.d`. In my case I also specified the user in the actual command of the crontab file, like so

```
* * * * * root env > /var/log/cron.log 2>&1
```

In this case we're writing the environment variables that cron sees into a log file. Once you redirect the output to cron.log you don't need to create that file explicitly. Once you've build and run your docker container, go inside the container and run `service cron status` to check of cron is running. If not, you can start it, for instance, by typing `cron`. The brings me to another important debugging technique: logging. It might be a good idea to install package `rsyslog` to Dockerfile, at least during the development phase (and if things to work out of the box). The creates system logs to which cron writes. Then, inside a running container, you can output cron's log message by

```
rsyslogd
tail --follow /var/log/syslog
```

Those log messages can contain valueable information if things to go as expected.

Lastly, we need to start cron upon container start:

```
CMD cron && my-awesome-executable
```

Using `jq`

You can always use `set x` in any bash script to show on standard out what the script executes. Here I use the `jq` (not installed by default) to substitute a value in a JSON file `config.json`. We first need to copy the command into a temporary file, and then move it to the actual file. Copy this line into a bash script and execute it with a desired value as argument. Use a temporary variable in order to do in-place editing.

```
tmp=$(mktemp)
jq --arg APP_NAME "$1" '.app_name |= $APP_NAME' .chalice/config.json > "$tmp" && mv "$tmp" .
```

Substitute a value for an existing key in a JSON file

Suppose you have `my_val="ABCD"` and you want to assign this as a value in an existing JSON file named `myfile.json`, which is an array at the first level.

```
tmp=$(mktemp)
jq --arg my_val "$my_val" '.[].my_key1.my_key2 = $my_val <path_to_JSON> > "$tmp" && mv "$tmp
```

Using `rsync`

Suppose we have a directory `dir1` on a remote machine and also local. To synchronize the content of that directory between the two locations, we run

```
rsync -avz user@machine:/home/user/dir1 dir1
```

The option `v` is verbose, `a` is ???

Control daemons

Get the status of a daemon like so

```
$ sudo systemctl | grep NAME
```

Control the status with

```
sudo service <daemon_name> <option>
```

where options are start or stop (among many others, see help file). You can also use

```
sudo systemctl NAME OPTION
```

where frequently used options are `start`, `status`, `stop`, `restart`.

AWS CLI commands

When working with AWS CLI we usually need to provide credentials. AWS' command-line utilty is `aws`, and `aws configure` can help setting up the credentials. Once set up they usually reside in `~/.aws/`.

To work with the `aws` utility set the profile configuration of your credentials into an environment variable. If the profile's name is `my-profile`, then run

```
export AWS_DEFAULT_PROFILE=my-profile
```

That way the `aws` utility can be used in a given shell without providing credentials with each command.

List files in bucket `bucket_name`, which has prefix `my_prefix`:

```
aws s3 ls s3://bucket_name/my_prefix/
```

Create a directory `my_new_dir` in a bucket `bucket_name`, which has prefix `my_prefix`:

```
aws s3api put-object --bucket bucket_name --key my_prefix/my_new_dir
```

Remove a file `somefile.txt` from a bucket `bucket_name`, with key `my_key`:

```
aws s3 rm s3://bucket_name/my_key/somefile.txt
```

Remove recursively `file1.txt`, `file2.txt`, `...` in prefix key `some_prefix` in a bucket

```
aws s3 rm s3://bucket_name/my_key/some_prefix/ --recursive
```

Careful, though, I think it also deletes prefix key `some_prefix`.

Installing Python-related bioinformatics graph tools

To create figures for the HCA project summary statistics I needed to generate t-Distributed Stochastic Neighbor Embedding (tSNE) figures, using the ScanPy package. This had two requirements: `louvain` and `igraph`.

I could install `igraph` using `pip install python-igraph`.

`louvain` is the implementation an algorithm of the same name, which maximizes modularity in graphs. Modularity here denotes groups, clusters of communities, so essentially a measure of similarity. The idea is that nodes which is similar to each other are connected by shorter edges than nodes that are different from each other. The algorithm is named by the work place of the researchers who discovered and published it (Blondel et al. (2008), namely the University of Louvain in Leuven, Belgium.

The Python package `louvain` contains a C implementation of that algorithm. But it has also igraph as a requirement.

Both packages are dependent on a number of libraries: `build-essential`, `libxml2-dev`, `libglpk-dev`, `libgmp3-dev`, `libblas-dev`, `liblapack-dev`, `libarpack2-dev`, `python-dev`. I installed those using `apt-get`.

Only after installing those librarries I was able to install

```
pip install louvain
```

Before I installed `louvain` I was able to use `python:3.6.9-slim-stretch` to dockerize it. But I'm not sure if I can install those libraries as they require the `apt-get` manager which I think that base image doesn't have.

Useful Docker commands

Note: most Docker commands need to be executed as *root* or using `sudo`. You can the username to the Docker group then you should be able to execute docker command without `sudo`.

List all Docker images on the system

```
docker images
```

List all Docker containers (i.e., built images) on the system

```
docker ps
```

Run a docker container `my-container` that contains a script that needs AWS credentials

We create a Docker volume with the `-v` flag, and set an environment variable using the `-e` flag.

```
docker run -v ~/.aws:/root/.aws -e AWS_DEFAULT_PROFILE=my-profile my-container
```

To start a bash shell to interact with docker container

```
sudo docker exec -it boardwalk_dcc-dashboard-service_1 bash
```

or `docker run --rm -it --entrypoint=/bin/bash 'name_of_docker_image'`

To tail a log file from a specific Docker container

```
sudo docker logs -f name_of_container
```

Stop containers from running

Use `docker stop <name-of-container>` to stop a running container (run `docker ps` to find out the name). Alternatively, you can use `docker kill <name-of-container>` (the former sends a SIGTERM followed by a SIGKILL, the latter sends a SIGKILL immediately).

Instead of `<name-of-container>` you can use command substitution `$(sudo docker ps -q)` (which prints a list of container IDs only) to loop over all containers. Identifying containers this waye can be used for any of the other docker commands in this paragraph.

Remove (delete) all containers

```
sudo docker rm $(sudo docker ps -a -q)
```

Executing this command has been useful in the case that a container (let's say *container1*) is used by another, stopped container (let's say *container2*). In such case, the image of *container1* can't be delete using the command below. Instead, when I executed it in that case, it returned

```
Error response from daemon: conflict: unable to delete <ID of container2> \
(must be forced) - image is being used by stopped container <ID of container1>
```

But if you run the above `rm` command the below `rmi` command should delete all images.

Remove (delete) all images from system (i.e., disk)

```
sudo docker rmi $(sudo docker images -q)
```

`docker-compose` commands

Stop and start Docker compose

`docker_compose1.yml` and `docker_compose2.yml` are Docker compose files.

```
sudo docker-compose -f docker_compose1.yml -f docker_compose2.yml down -v
```

If you get

```
sudo docker-compose -f dev.yml down -v
ERROR: build path /home/ubuntu/cgp-deployment/boardwalk/boardwalk either does not exist, is
```

it means that you had not brought up the containers using `dev.yml`.

Reading the logs for debugging

For development work reading the logs is very useful. For instance, the Docker compose network of containers of CGP's computation platform consists of 9 containers, each of which has a log.

```
docker exec -it core-nginx bash
```

Use

```
sudo docker-compose logs
```

That command only accepts the file names `docker-compose.yml` or `docker-compose.yaml`. So if your config files are named differently you need to copy them to one of the two names in order to be able to read the logs.

```
sudo docker-compose -f docker_compose1.yml -f docker_compose2.yml up -b
```

Also in development mode, if you want to built the Boardwalk containers from the image, then first bring all containters in `boardwalk/` down

```
sudo docker-compose -f dev.yml down -v
```

which also removes all volumes from the network. Then remove the images needed for Boardwalk

```
sudo docker rmi <image_id>
```

There should be 3-4 images. Then bring up the network by using

```
sudo docker-compose -f dev.yml --build
```

That will build the containers from the images.

Delete containers from system

Purge (i.e., delete) all unused or dangling images, containers, volumes, and networks

```
docker system prune
```

Use it with the `-a` option to remove any stopped container and all unused images (not just dangling). You'll be surprised how many deleted lines it prints... At the end it reports the amount of reclaimed disk space.

After the command has finished verify that no image is left by `sudo docker images`. It should show no entries.

Other Docker commands

(all docker commands need to be prefixed with `sudo`)

```
# remove exited containers:
docker ps --filter status=dead --filter status=exited -aq | xargs -r docker rm -v

# remove unused images:
docker images --no-trunc | grep '<none>' | awk '{ print $3 }' |
   \ xargs -r docker rmi <image_id>

# remove any image (run docker images first to get the IMAGE ID, suppose it's 60503d5b81fb)
docker images | grep 60503d5b81fb | awk '{print $1 ":" $2}' | xargs docker rmi

# remove unused volumes
docker volume ls -qf dangling=true | xargs -r docker volume rm
```

Problem with a native `nginx` service running on my VM

First find out whether a native (i.e., not dockerized) `nginx` is running

```
sudo service nginx status
sudo service nginx stop
```

and stop its service. Next find out

```
sudo docker run -i -t 2e92148b6758 "nginx -g 'daemon off;'"
```

How to test webpage functionality of WSGIs

If anything goes wrong with a webservice, the first thing to check are the logs. You need to log in as root to read it. On a VM do

```
sudo -i
less dcc-dashboard-service/logs/error_uwsgiB.log
```

If you execute `G` (a VI command to get to the last page) you can move around the text page-by-page.

When programming the backend to affect actions controlled by a frontend webpage it is most useful for testing to monitor the logs. For instance, when I worked on the manifest handover I checked what happened to the code on the backhand by monitoring `~/dcc-dashboard-service/logs/error_uwsgiB.log` as root and running it in the foreground (`-f` flag) by using the `tail` command:

```
~/dcc-dashboard-service/logs# tail -f error_uwsgiB.log
```

If you operate some functionality on the webpage (i.e., click a button) you see immediately how info messages and error messages fly by on the terminal. Use `Ctrl-C` to exit.

`pip` installations

You want to uninstall all packages in the current virtual environment

```
pip uninstall -y -r <(pip freeze)
```

`venv` command in Python 3: the recommended way to create virtual environments

First install the module

```
sudo apt install python3-venv
```

To create a virtual environment `newenv` use

```
python3 -m venv newvenv
```

I'm not sure how to create a virtual environment using a specific Python point version as is possible to do using `virtualenv` (here is my question on Stackoverflow).

Internet speedtest

Download a pip-installable app:

```
pip install speedtest-cli
```

Then run

```
speedtest-cli
```

and it will report download and upload speed.

How to secure copy data to and from an EC2 instance

```
scp -i "/home/michael/.ssh/sec_key_mk.pem" file_to_scp.txt ubuntu@ec2-35-161-55-66.us-west-2
```

Install `umake`

```
sudo add-apt-repository ppa:ubuntu-desktop/ubuntu-make
sudo apt update
sudo apt install ubuntu-make
```

and that's it. Run `umake` on the command line to see the options. I've used it to install Eclipse.

Java

IntelliJ

I installed the community edition (CE) by using *snap*, which is nice as it updates automatically to the latest version.

Find libraries and install them

When IntelliJ is installed using *snap*, find all libraries in

`/snap/intellij-idea-community/current/lib`

In case you lost the project structure

File -> Projectstructure -> modules -> import module should solve it.

Install Eclipse

I've used `umake` (see *umake*) to install the latest version of Eclipse. After `umake` is installed, it's a breeze. The following install Eclipse for use with `Java`.

`umake ide eclipse`

and done.

How to `scp` to a AWS AMI

```
scp -i "/home/michael/.ssh/sec_key_mk.pem" file_to_scp.txt ubuntu@ec2-35-161-55-66.us-west-2
```

Use a start-up script to stop the avahi daemon

The avahi daemon seems to cause a problem after each reboot. I decided to write a startup script that stops its service right during reboot. This need to work with `systemd`, a new system introduced with Ubuntu 16.

To start a service we need two files: 1. a shell script `stop-avahi` in `~/bin` 2. a service script `stop-avahi.service` in `/etc/systemd/system`

The shell script has the content:

```
#!/bin/sh

sudo rm -rf /var/crash/.lock*
sudo rm -rf /var/crash/*
sudo service avahi-daemon stop
```

and the systemd script contains:

```
[Unit]
Description=Power-off gpu

[Service]
Type=oneshot
ExecStart=/usr/bin/vgaoff

[Install]
WantedBy=multi-user.target
```

I made both scripts executable with `chmod +x ...`, and then executed

```
$ systemctl enable stop-avahi.service
Created symlink from /etc/systemd/system/multi-user.target.wants/stop-avahi.service to /etc/
```

which started the service. It should start automatically after reboot. See also this link

Setting the ALPS Dual Trackpoint on Dell Precision 7510 laptop running Linux

Settings for the trackpoint are not set right by default. It is oversensitive such that I was unable to get the pointer into any small spot. In order to set it on Linux (Ubuntu) we first need list the identifier for the mouse and track pointer components:

```
xinput list
```

It shows that the AlpsPS2/2 ALPS DualPoint Stick has ID 15. Another interesting list of settings is returned by

```
xset q | grep -iA 1 pointer
```

although the interpretation of these settings are not yet clear to me. But moving on, we now want to know what are the properties of the pointer stick, and we find out by querying it:

```
xinput --list-props 15
```

The parameters of importance are *Device Accel Constant Deceleration (295)*, *Device Accel Adaptive Deceleration (296)* and *Device Accel Velocity Scaling (297)*. I've set these values to 5, 5, and 10, respectively, using

```
xinput --set-prop 15 295 5
xinput --set-prop 15 296 5
xinput --set-prop 15 297 10
```

Further, to adjust the scrolling of the page using the stick point while holding the middle button I set

```
xinput --set-prop 15 348 100
```

Following this bug-link I create a file named `71-pointingstick-precision.hwdb` with content `evdev:name:*DualPoint Stick:dmi:bvn*:bvr*:bd*:svnDellInc.:pnPrecision7510*:pvr* POINTINGSTICK_CONST_ACCEL=0.025`, although I do not understand the impact of it (i.e., this value is very different from the one I input above for the contstant acceleration). After this I executed

```
sudo systemd-hwdb update
sudo udevadm trigger
```

Lastly, I create a file `.xsession`, in which I wrote `xinput set-prop 'AlpsPS/2 ALPS DualPoint Stick' 'libinput Accel Speed' 1`.

Before I did the last step the step before the last step I noticed no changed when I changed parameters in the `xinput`, and only after these two last steps did such changes take effect.

GUI

Top bar of window is off screen after plugging laptop back docking station

`ALT + spacebar`, then choose `move` and recenter it

Working with Python 3

Install

```
sudo apt-get install -y python3-pip
```

then create a virtual environment `.envblue`

```
virtualenv --python=python3 .envblue
```

or like so

```
virtualenv --python=$(which python3) .envblue
```

and `source` it, or

```
virtualenv --python=python3.6 .venv
source .venv/bin/activate
ipython3
```

to use Python 3.6 in your IPython (Python 3.6 needs to be installed on your system).

```
pip3 install python
pip3 install ipython
```

Batch install all modules listed in `requirements.txt`

```
pip3 install -r requirements.txt
```

in Python 2.7 that should work (unconfirmed)

```
pip install -r requirements.txt --no-index --find-links file:///tmp/packages
```

Installing nvidia graphics driver on the Dell Precision 7510 laptop

This manual should lead to an easy installation of Nvidia drivers for the graphics card. This is much easier compared to the installation on Debian, which does not have a graphics card PPA (*personal package archive*), and where I need to install the run file I had downloaded from Nvidia.

1. Find out what graphics

Identify the graphics card(s) using

```
$ sudo update-pciids
$ lspci -nn | grep -i --color "2d\|3d\|display\|vga"
```

This will return two graphics cards, one Intel, one Nvidia. The Intel I think comes with the motherboard, the Nvidia GM107GLM Quadro M2000M I ordered with the system, but it isn't used after a fresh Ubuntu installation. You need to get the Nvidia drivers for it and install them. For Ubuntu 16.04 this seems easiest by using apt-get.

For the remainder of the installation I followed these instructions

2. Add the graphics driver PPA as shown here

```
sudo add-apt-repository ppa:graphics-drivers/ppa
sudo apt-get update
```

I think this is `deadsnakes-ubuntu-ppa-xenial` in `/etc/apt/sources.list.d/`.

3. Now simply use apt-get to install the driver. I did

   ```
   $ sudo apt-get install nvidia-352
   ```

   and rebooted the machine. Everything went just find, no black TTY log-in screen, my desktop manager ran just as before. According to Nvidia my card is compatible with driver version 352 (http://www.nvidia.com/Download/driverResults.aspx/95159/en-us).

4. After reboot I checked the Nvidia GUI. The installed driver version is 384, and everything seems to work. It shows that the card has 4 GB of memory, and can also save configurations.

5. Check information on Nvidia driver Run `$ nvidia-smi`. This will list all features of your installed Nvidia card. Also, `lsmod | grep nvidia` will now have an output, while `lsmod | grep nouveau` will have no output (it should have had an output before the installation).

6. The only way to prevent the system from upgrading (I think this could become a problem when I use specific GPU settings, which rely on the version), we need to uninstall the graphics driver PPA.

   ```
   $ sudo add-apt-repository --remove ppa:PPA_Name/ppa
   ```

   where PPA_name/ppa is `graphics-drivers/ppa`.

7. If you want to uninstall the Nvidia driver use `sudo apt-get purge nvidia*`.

SSL certificates

The purpose of a certificate is to certify to the browser that if you point it at bankofamerica.com you indeed handshake with the website of the Bank of America, and not some imposter website. This is done by a third-party certificate authority (CA) which has issued BoA a certificate, which works by private / public key encryption.

In development, for example when using letsencrypt, fake certificates can be used instead of real certificates.

`ssh` issues

`Offending key for IP in /home/<username>/.ssh/known_hosts:<line_num>`

This can happen, for example, if you have two EC2 instances, say *I1* and *I2*. You logged in to *I1* just find using `ssh`. Then you dissociate its *Elastic IP* and associate it with *I2*. Once you try to `ssh` into *I2* the private / public key handshake doesn't work anymore (because it's not the one from *I1*), but the IP is correct. That's when you see this line.

You can remedy this using brute force by just deleting that line in file *known_hosts* using

```
sed -i '<line_num> ~/.ssh/known_hosts
```

assuming the *line_num* is the line number shown in the warning.

`xclip`

If you have a password saved in a file, and you want to copy it to the X11 clipboard (i.e., without displaying it on the screen), you can use `xclip` like so:

```
xclip -sel c < file
```

where `file` is the file that contains the password.

AWS

Key for `*.pem` file

It's in the *Secrets Manager*, and there click the button *keys*

Checking log files in *CloudWatch*

This is an example from repo *azul*, where we want to check the logs of the AWS lambdas and others. In the AWS console go to *CloudWatch*, then in *Logs*. There use the search prompt. To find the logs for the service lambda in the *dev* deployment search for `/aws/apigateway/azul-service`. To find the logs for the service lambda running my personal deployment, search for `/aws/lambda/azul-service-mjkrause`

Note that time in the *Logs* page is reported as local time. But if you click on a specific log stream it is reported as UTC (should be 7 h difference).

In the upper left corner, right above the table of Time/Message, click on the timestamp. A window opens which can be used to set the time interval of log

times. Of note is the *Relative* tab, which can be used to return only logs from the last say 10 min. It also has the option between local and UTC time. It sometimes takes awhile for the log to appear. Click on the refresh button often.

Perl one-liners

Use regex to extract specific lines from a text file.

Suppose we want all lines which start with exactly two whitespaces (e.g., this could be some configuration file).

```
in_file=$( perl -ne 'print if /^\s{2}(?!\s)/' filename.txt)
```

JavaScript, Node.js, and TypeScript

Developing inside a running Docker container

This set of instructions is specific to developing an app created with Strongloop's Loopback.

Suppose you have a Docker container or a Docker-compose script which launches multiple containers, and you would like to develop the code-base while your code is running inside the container (and the container(s) could run on localhost or remotely). This can be done with VS Code.

1. Launch VS Code (`code .`) inside the directory that contains the code-base opens the IDE. Then click the extensions icon, and and find and install the `Remote Explorer` extension.
2. Navigate to the directory containing the code-base, including the `Dockerfile` or the `docker-compose` file. Launch the container(s).
3. In VS Code, click on the Remote Explorer icon (on the left side of the window by default). You should see the container(s) listed. Pick the one that contains the code you want to develop, and connect to it as described in the next step.
4. Hover over the container of interest. Use the right mouse button and click on `Connect to Container`. That action opens another VS Code editor which shows the content of the container. Navigate to the directory holding the code. Also, you should see a green bar on the lower left side of the screen that say should something like `Container ...` describing the name of the container.

Now you're connected to the container, and can edit the code inside the running container. Upon saving the file in which we made a change it should automatically re-compile the code, and effects should take effect immediately. In case of a Loopback/Angular app go to `localhost:4200`.

View the log-file as it compiles

Go to the Remote Explorer (left side panel in GUI). It should show all containers. Hover over the container that is attached and in which you want to edit the code. Click on it with the right mouse button and select `Show container log`. In console (at the bottom of the GUI) click on tab *Terminal*. Now go to the upper right corner of the terminal. There should be a pull-down menu. Select the name of the container. The terminal should now show the output of the log file of the running container. So if you edit code and hit *save* it should show that the code recompiles.

Debugging Node.js code

Suppose you have a Node.js application running as a backend to some frontend. You would like to exercise some code in the backend that is called by some HTTP method in the front end.

1. Set a breakpoint at some line where you want the debugger to stop. Next hit `F5` or navigate to menu `Debug` -> `Start Debugging`. In the debugger tab you should see that the debugger has attached to some process.
2. Identify the endpoint that invokes the Node.js backend method and create a HTTP request. In a terminal run `curl <METHOD> http://myendpoint.org`.
3. The debugger should stop at the first breakpoint it passes, i.e., the red dot should be framed with a yellow arrow. Use `F10` and other keys to inspect variables and call stack. Et voila...