Git cheat sheet

Efim Abrikosov

June 23, 2020

1 Main commands

1.1 Git

- `git help` *commandname* — display help information about Git command
- `git init` *[--template=templatedirectory]* — create an empty Git repository or reinitialize an existing one. Files and directories in the template directory whose name do not start with a dot will be copied to the directory after it is created.
- `git clone` *repository [directory]* — clone a repository into a new directory. Optionally supply the name of a new directory to clone into.
- `git remote add` *name url* — add a remote (e.g. named "origin") for the repository at a specified web address
- `git add` *[filename]* — updates the index using the current content found in the working tree, to prepare the content staged for the next commit. If filenames are specified, then only given files are staged
- `git commit` *[-a][-m text]* — record changes to the repository
- `git diff` *[cached]*
- `git status`

1.2 Anaconda environment management

- `conda commandname -h` — display help information about command
- `conda env list` — list all anaconda environments
- `conda list -n myenv` — list packages installed in the environment
- `conda activate [envname]` — activate anaconda environment
- `conda deactivate` — deactivate current environment
- `conda create [--clone envname] -n newenvname` — create a new environment
- `conda remove --name myenv --all` — remove existing environment
- `conda install [--name myenv] packagename [= version]` — install package, if no environment is specified, the package is installed in the current environment
- `conda search packagename` — search for packages with a given name
- `install --revision revnum` — revert to a given revision
- `conda remove [-n myenv] packagename` — remove package, if no environment is specified, the package is removed from the current environment

1.3 Base workflow cases

2 Setting up a Google Cloud project

- Log in your google account
- Browse to cloud.google.com
- Click on “Go to Console”
- Go to Navigation menu (three horizontal lines in the top left corner)

- Select “Compute Engine”
- Click “Create”
- Select “Allow full access to all Cloud APIs”
- Now click “Create”
- Click on “SSH” field in the VM list to open the console
- To update the system configuration type “sudo apt-get update”
- To install Git type in “sudo apt-get -y -qq install git”
- Go to Navigation menu
- Select “Storage”
- Click “Create bucket”
- Select appropriate settings
- Now click “Create”
- In console type gsutil cp [filename] gs:://[bucketname]/[pathname]
- To publish cloud storage files to the web run gsutil acl ch -u AllUsers:R gs://[bucketname]/[pathname]
- To launch Cloud Datalab, open a Cloud Shell in the Platform page (the icon is in the top right corner)
- In Cloud Shell type “gcloud compute zones list”
- In Cloud Shell type “datalab create mydatalabvm –zone [zonename]”
- Creating Datalab VM may take several minutes
- Click on “Web Preview” button in the top of the Cloud Shell and change port to 8081
- Go to Navigation menu
- Select “BigQuery”

- In More Options click “Query settings”
- Under Additional Settings ensure that Legacy is not enabled
- In the query textbox type necessary SQL commands to extract data from big datasets
- Create a notebook in Datalab
- Define a valid query string in the notebook
- Use the following logic:
    1. import google.datalab.bigquery as bq
    2. df = bq.Query(query).execute().result().to_dataframe()
    3. df.head()
- Launch Cloud Datalab
- To download git repository contents use the logic
    1. %bash
    2. git clone [repositoryaddress] m -rf [pathname]
- Select APIs&Services from Cloud Platform Navigation Menu
- Click “Library” and search for required API (e.g. Cloud Vision API, Translate API, Speech API, or Natural Language API)
- Click “Enable” if necessary
- In APIs&Services click “Credentials” and create “API key” if necessary. This key will be used in Datalab code to invoke various APIs
- In Datalab use APIKEY generated in credentials as “developerKey” parameter in Datalab code
- In Datalab, run “!pip install –upgrade google-api-python-client”
    - Translate API
        1. from googleapiclient.discovery import build

    2. service = build('translate', 'v2', developerKey=APIKEY)
  - Vision API
    1. import base64
    2. vservice = build('vision', 'v1', developerKey=APIKEY)
  - Natural Language API
    1. lservice = build('language', 'v1beta1', developerKey=APIKEY)
  - Speech API
    1. build('speech', 'v1beta1', developerKey=APIKEY)

- Launch Cloud Datalab
- Use the following logic
  - google.datalab.bigquery as bq
  - qry = ''' SELECT * FROM ...'''
  - bq.Query(qry).execute().result().to_dataframe()

2.1 Setting up a new Google Compute Engine VM

Better, follow instructions here.

Make sure that suitable python version is installed: `python --version` `python3 --version`

If python or python3 is not installed on the machine, follow instructions here. Note that python version must be compatible with your distribution system (Ubuntu/Debian/etc...)

Install 'pip' package management system with distribution: `sudo apt install python3-pip`

Check that pip is installed `python3 -m pip --version`

Install 'virtualenv' package. WARNING: for now do not simply use `pip3 install [package name]`, rather rely on `python -m pip install [package name] –user`. This ensures that package management is controlled by pip and not by OS distribution system ()some details). `wh`

2.2 Using Google Cloud AI Platform to train models

Use your local environment to develop a model. Organize it in the form of a python package with *task_script.py* script that can be launched locally using the following minimal example

```
python -m task_package.task_script /
  --job-dir <path to job dir> [optional arguments]
```

Typically the task script will contain instructions to parse command line arguments, define a model and data pipeline, train the model record checkpoints and finally store model weights,

When the package is ready it can be seamlessly transferred to run on Google Cloud AI Platform with the following minimal example:

```
gcloud ai-platform jobs submit training $JOB_NAME /
  --package-path $PATH_TO_PACKAGE /
  --module-name task_script /
  --region $REGION /
  --job-dir $JOB_DIR /
  --stream-logs /
  -- [optional arguments accepted by python task script]
```

Supplied arguments are as follows:

- `$JOB_NAME` unique name assigned to a job. List of jobs can be viewd with `gcloud ai-platform jobs list`
- `--package-path` local path to the package (e.g. `task_package/`)
- `--module-name` name of task script in the package (e.g. `task_script`)
- `--region` name of gcloud compute region (e.g. `us-east1`). List of regions can be viewd with `gcloud compute regions list`
- `$JOB_DIR` valid path accessible by AI Platform. For instance, can use a directory in a Google Cloud Storage Bucket (list project buckets with `gsutil ls`)
- `--stream-logs` view script stdout in command line during the execution

Further control over execution can be fine-tuned with the following useful AI Platform options:

- `python-version 3.7`
- `--runtime-version 2.1` use TensorFlow version 2.1 for execution (list of available runtime versions and packages within)

3 Useful links

- Git cheet sheet
- Anaconda command list