|  |  |  |  |
| --- | --- | --- | --- |
|  | git init <repo-name>  git init OR git init <path/to/project>  git clone <path> <name>  git remote -v  git remote add <remote-name> <URL>  git remote rm <remote-name> | create new git repository in current dir  convert current dir/path to git repository  create copy of project, can name project (optional)  show remote repository URL  add remote to point to URL  remove existing remote | |
|  | git status  git show <hash>  git annotate <filename>  git ls-tree -r --name-only HEAD  git fetch --all  git ls-remote  git status -uno | show which files are in staging area, and which files have changes that haven't been added  can just input 1st 6 hash characters, to view commit, or HEAD, or HEAD~2  see which author edit which line of file in which commit  list all files currently in repository being tracked  list all branches on remote repo  list all branches on remote repo  git status only for tracked files | |
|  | git add <filename>  git reset HEAD  git reset HEAD <filename>  git checkout -- <filename>  git checkout -- .  git checkout <hash> -- <filename>  git rm <filename>  git rm --cached <filename> | add file to staging area  unstage files in staging area  unstage particular file in staging area; can combine w git checkout to undo changes in staged files  discard changes made in files that are not staged yet  discard all changes in repo  restore particular file to previous commit according to hash  removes files and stage removal of file  stop tracking file, file still in working directory | |
|  | git diff <filename>  git diff  git diff <directory>  git diff -r HEAD  git diff -r HEAD <path/to/file>  git diff <hash>..<hash>  git diff <branch1>..<branch2>  git diff <branch> --name-only  git diff <branch> -- <filename> | compare current file to previous last saved  show all changes in repository  show changes to file in particular directory  -r: compare to a particular revision, HEAD: most recent commit  compare 1 particular file, HEAD~1: 2nd last most recent commit, ... so on  compare btw 2 commits; OR git diff HEAD~1..HEAD~4  compare btw 2 branches  names of files w changes from branch compared to current branch  diff in the file btw branch and curr branch | |
|  | git commit -m "message"  git commit --amend -m "new message"  git commit | save everything in staging area  change commit message  open text editor to write a longer commit message | |
|  | git log  git log <path>  git log -3 --name-only  git log -5 --stat  git log -3 -p  git log --since="1 day"  git log --after="yyyy-mm-dd hh:mm:ss"  git log --author="author"  git log --oneline --decorate --graph --all  git log --reverse  git log --help  git log -S<search> | log of project's history, most recent shown first (press space to go to next page, 'q' to quit)  log of path  3 most recent commit, name of files only  5 most recent commit, num of lines changed in files  show difference in lines  time optional  show commits in reverse order  (/: to search; n: next search; q: quit)  search for "search" in all commits in repository | |
|  | git branch  git switch <branch>  git switch -c <branch>  git push --set-upstream origin <branch> | see what branches are there  switch to another branch  creates new branch and switch to it  allow github to track new branch | |
|  | git push origin <branch>  git pull origin <branch> | (branch optional)  does git fetch + git merge, (branch optional) | |
| merge conflict | git merge --abort  git merge <source>  OR  git add <edited file> | return to state just before you pull  merge from source to current branch  edit file, (<< from local version, == divides the diff btw diff versions, >> from remote version) | |
| git diff output | diff --git a/report.txt b/report.txt  index e713b17..4c0742a 100644  --- a/report.txt  +++ b/report.txt  @@ -1,2 +1,2 @@  -# Seasonal Dental Surgeries 2017-18  +# Seasonal Dental Surgeries (2017) 2017-18  # TODO: write new summary | | (diff --git): command used, a and b are placeholders = 1st and 2nd version  denote "-" represent a/report.txt  denote "+" represent b/report.txt  -1: a/report.txt from line 1; 2: 2 line from a; +1: b from line 1; 2: 2 line from b  line removed  line added  line no change |
|  | file1  \*.py  git clean -n  git clean -f | create file .gitignore to tell git which files you don't want to save  #ignore file1, ignore files ending with .py  show files in repository not tracked by git  delete files not tracked | |
|  | git config --list  --local OR --global OR --system  git config --global <setting> <value> | see git configurations  see specific git configs (optional)  set configurations for setting | |
|  | git blame <filename>  git blame -L <linefrom>,<lineto> <file> | see who modify the file  see who modify lines in file | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | docker run <image>  docker run --name <name> <image>  docker run --name <name> -d -p <hostport:containerport> <image>  docker ps -a  docker image ls  docker logs <container> | | | | | | | name a container and run it  -d: return control back to terminal; -p: port  show all containers  list all images  view logs of container |
|  | docker start -a <container>  docker stop <container>  docker restart <container>  docker rm <container>  docker image rm <container> | | | | | | |  |
|  | docker exec -it <container> /bin/bash  cd etc  cd nginx/conf.d/  cat default.conf  apt update  apt install -y nano  cd /usr/share/nginx/html  exit  cd <some directory>  docker cp <some page>.html <container>:/usr/share/nginx/html  docker run --name <name> -d -v <local-dir>:/usr/share/nginx/html -p 8080:80 nginx | | | connect to a running container (-it: run interactively, as if u are in the container & open bash shell)  debian\_version file present in etc folder -> so we know the os  show where html file is located and displayed  install nano, no need to type Y  where html files are located (found in default.conf)  exit bash shell  go to some directory on local computer that contains html file  copy html file from local computer to container directory  mount folder on local drive to folder in container | | | | |
| Linux | /bin  /etc  /home  /lib  /mnt  /root  /tmp  /usr  /var | | binary files for system, system admin and users  most impt system config files  home directories of common users  shared library files  standard mount point for external file systems  admin user home directory  temporary space, cleaned upon reboot  programs, libraries, documentation ... for user related programs  storage for all variable files and temp files created by users | | | | | |
|  | RPM-based: use sudo yum install xxx  Debian-based: use sudo apt install xxx | | | | | | |  |
|  | source bin/activate  ###inside python file. app = Flask(\_\_name\_\_)  ###@app.route('/handler')  ###def ...: lang = request.args.get('value')  export FLASK\_APP=hello  flask run  #now go to localhost:5000/handler?value=hi  CTRL+C  deactivate | | | | | | | activate python virtual environment  uses name of python file (e.g. hello for hello.py)  decorater  send hi to value in handler in flask app  stop flask app  deactive virtual env |
| Docker-file | FROM <image>  WORKDIR /app  COPY <file on local> <file in container>  RUN pip3 install -r requirements.txt  ENV FLASK\_APP hello  ENTRYPOINT ["python", "-u", "-m", "flask", "run", "--host=0.0.0.0"] | | | | | | | -built start-  create and set wd in container  COPY . . (to copy everything in local directory)  -built end-  execute when container runs (-u: don't buffer output, -m: run module)  use either ENTRYPOINT or CMD |
| Docker build | docker build -t <tag> . | | | | | | | build from Dockerfile in . (i.e. current dir) |
| Docker Compose | | docker compose up -d  #docker compose restart  docker compose down | | | | | To spin up multiple container at once  To stop the network and remove running containers | |
|  | | @app.route("/")  def fn(): | | | | function to call when request to website path/ is sent | | |
| Docker yml | | version: "3.9"  services:  web:  build: .  ports:  - "8000:5000"  redis:  image: "redis:alpine" | | | version of yaml file  web container  build web container from Dockerfile located in current dir  port 8000 on local machine map to 5000 of web container  redis container  build redis container from image | | | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | cp <oldfile> <newfile>  mv <oldfile> <newfile>  rm <file>  rmdir <name>  mkdir <name>  cat <file>  head <file>  head -n <num> <file>  ls -R  ls -R -F  man <command>  cut -f <num1>-<num2>,<num3> -d <delimiter> <file> | | copy file  move file  remove file  remove directory  create directory  concatenate file contents  show 1st 10 lines of file  show 1st <num> lines of file  list all files in current and children directory  -F: print a '/' after name of every directory and '\*' after name of runnable program  help page for command  -f: get col btw num1-num2 & num3; -d: specify delimiter |
|  | less <file1> <file2> | | 'less' paginate file output, :n next file, :p previous file, :q, quit, 'spacebar' page down |
|  | grep <term> <file>  -c  -h  -i  -l  -n  -v | | print all lines in file containing term  print count of matching lines  do not print names of files when searching multiple files  ignore upper, lower casing  print names of files that contain matches, not the matches  print line numbers for matching lines  invert the match (i.e. show lines that don't match) |
|  | <some command> > <file>  <some command> | <other command> | | save output in file  pipe output from some command into other command |
|  | wc  uniq | | print num of **c**haracter, **w**ords and **l**ines in a file (use -c, -w, -l respectively)  remove adjacent duplicate lines (-c to display counts) |
|  | \*  ?  []  {} | | wildcard, ≥ 0 times  wildcard, 1 time  matches any character in bracket  matches any of the comma-separated pattern in bracket (e.g. {\*.csv, \*.txt}] |
|  | sort  -n, -r, -b, -f | default ascending alphabetical order  by numerical order, reverse order, ignore leading blanks, fold case (i.e. case insensitive) | |
|  | HOME, PWD, SHELL, USER, OSTYPE  echo $HOME | | enviroment variables  print value of environment variables |
|  | <variable>=<value>  for <variable> in <loop>; do <body>; done | | shell variable (like local var), assign value to variable  for loop |
|  | bash <file>  $@  $1, $2 | | run file  replace w all command line params given to the shell script  1st command line param, 2nd command line param |

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP requests | URL + Method (GET, POST, PUT, DELETE) + Headers (meta-data) + Body | | #POST: create new resource; PUT: update/edit existing resource |
| HTTP response | Status code + Headers + Body | | Status code: 404: not found; 200: successful request; 201: successfuly created resource |
| Python requests | import requests, from datetime import datetime  req\_out = requests.get(url, params = params1)  req\_out.json() | params should be in python dict  e.g. param1 = {'date\_time': dt.strftime(datetime.now())}  request.post(url, files=files) | |
|  | from sklearn.preprocessing import OneHotEncoder  from sklearn.compose import ColumnTransformer  ct1 = ColumnTransformer(...)  ct1.fit(X); X\_new = ct1.transform(X)  lm1.fit(X\_new, y\_train)  import joblib  joblib.dump((lm1, ct1, df), 'filename.pkl', compress=3)  (lm1, ct1, df) = joblib.load() | | #tranformer model  #fitted model  compress models & df into 1 file  load back models and df |
|  | from flask import Flask, request, jsonify, send\_file, render\_template  @app.route('/', methods = ['GET'])  def fn():  var1 = request.args.get('arg1')  return jsonify()  @app.route('/plot', methods = ['GET'])  def fn():  return send\_file('img.png')  @app.route('/upload', methods = ['POST'])  def fn():  return render\_template('uploaded.html', fname = f.filename) | | #for get request send to url/  #obtain params send in get request  #return as json string  #for user to upload files to url/upload  #jinja template?? |
|  |  | |  |
|  |  | |  |
|  |  | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Plotly | | import plotly.express as px  line\_graph = px.line(dataframe = df, x = "", y = "", title = "", color = "")  line\_graph.show()  bar\_graph = px.bar(dataframe = df, x="", y="", title = "", orientation = "')  bar\_graph.update\_layout({'parameter': value}) | | | | | | | | | color optional  orientation either h or v  update parameters of graph | |
| Dash | | import dash  import dash\_core\_components as dcc  app = dash.Dash()  app.layout = dcc.Graph(id = "example-graph", figure = bar\_fig)  if \_\_name\_\_ == 'main':  app.run\_server(debug=True) | | | | | | | | | initialise app  create layout  need run *python my\_app.py* in command line | |
|  | | import dash\_html\_components as html  app.layout = html.Div(children=[  html.Div(style = {'background-color': 'red', 'height' = '250', 'width' = 250}),  html.Div(children=[html.H1('This box'),  dcc.Graph(id = "example-graph", figure = bar\_fig)  html.H6('Another title')],  style = {'background-color': 'lightblue', 'height' = '250', 'width' = 250}) ]) | | | | | | | | | 1st box  2nd box  graph in 2nd box | |
| html | | html.Br()  html.Img(src='')  html.Ul()  html.Ol(children=[html.Li(children=[x, 'and', y]), html.Li(children=[a, 'and', b])])  html.Li() | | | | | | | | | new line break  insert image  unordered list, bullet point  ordered list, numbered point  each list element | |
|  | | html.P() OR  html.Span(children=[f"Date: {datetime.now().date()}", " by ", html.B('Author')])  html.B()  html.I() | | | | | | | | | insert plain text, accept children args  insert plain text, accept children args  bold text  italicize text | |
| CSS | style = {'background-color': 'red', 'color' = 'rgb(0,0,255)', 'height' = '250px', 'width' = 50%}) | | | | | | | | | | color: text color. Both color accept strings or rgb | |
|  | | | 'border': '<width in pixels> <style (dotted/solid)> <color>'. e.g. 'border': '4px solid black'  Specify 4 num for each property (padding & margin): from top, right, bottom, left  OR specify 1 num to applied to all sides OR specify 2 nums to apply to top-bottom and left-right  'padding': '10px 5px 5px 15px' OR 'margin': '100px auto'  Content: size using 'inline-size', 'block-size', 'width' and 'height'  Inline-block: can set height/width and can be side-by-side. 'display': 'inline-block' | | | | | | | | |
| Callbacks & Dropdowns | | | from dash.dependencies import Input, Output  app.layout = html.Div(children=[  dcc.Dropdown(id='title\_dd',  options=[{'label': 'Title 1', 'value': 'Title 1'},  {'label': 'Title 2', 'value': 'Title 2'}]),  dcc.Graph(id='my\_graph')])  @app.callback(  Output(component\_id = 'my\_graph', component\_property='figure'),  Input(component\_id='title\_dd', component\_property='value')  )  def triggered\_function(data):  return fig | | | | | | label is shown to user, value is what is passed to callback  # id: identify component; property: what will be changed  #output from here is send to app  #input from app get send here  #function for output in dcc.Graph | | | |
| Interactive Component | | | dcc.Checklist()  dcc.RadioItems()  dcc.Slider(min=10, max=50, value=45, step=5, vertical=False) OR dcc.RangeSlider()  dcc.DatePickerSingle(date=date(2023,1,27), initial\_visible\_month=datetime.now()) OR  dcc.DatePickerRange(initial\_visible\_month=datetime.now(), start\_date=date(2023,1,27), end\_date=date(2023,1,31))  ### All need an id param for input selection to be sent to app | | | | | | | Checkboxes  Radio buttons  Slider selectors. value: default value  Date selector. date: default date. Optionally can set min\_date\_allowed & max\_date\_allowed  Select range of dates  ### component\_property can be date or value | | |
| dcc.Input(id='my\_input', type='text', placeholder='Enter text')  #number: can set min and max  #text: can set minLength/maxLength, pattern (for regex)  default type is text, #type will auto set validation | | | type: number, password, email, range, tel, url, search, hidden)  disabled=True param to disable input or  required=True param to force user input  debounce=False (callback as you type; default is callback on unfocus or 'Enter') | | | | | | |
| Hover and on-click | | html.Div(children=[  dcc.Graph(id='bar\_fig', figure=ecom\_bar),  html.P(id='text\_output')])  @app.callback(  Output('text\_output', 'children'), #output based on hover over graph  Input('bar\_fig', 'hoverData')) #hovering over graph changes output | | | | | @app.callback(  Output('text\_output', 'children'),  Input('bar\_fig', 'clickData')) #clicking change output | | | | | |
| Chained callback | | @app.callback(  Output('minor\_cat\_dd', 'options'), #input from major dd update  Input('major\_cat\_dd', 'value')) #option in minor dd  def update\_dd(major\_cat\_dd):  return minor\_options  @app.callback(  Output('minor\_cat\_dd', 'value'), #input from minor dd option  Input('minor\_cat\_dd', 'options')) #update values shown in minor dd  def update\_dd(minor\_cat\_options): | | | | | | #for multiple outputs  @app.callback(  Output('my\_title', 'children'), #title\_value sent here  Output('minor\_cat\_dd', 'options'), #dd\_value sent here  Input('major\_cat\_dd', 'value'))  )  def some\_function(input);  return title\_value, dd\_value | | | | |
| Table | | from dash\_table import DataTable, FormatTemplate  money\_format = FormatTemplate.money(2)  d\_columns = [{'name': 'Major Category', 'id'; 'Major Category'},  {'name': 'Total Sales ($)', 'id'; 'Total Sales ($)', 'type': 'numeric', 'format': money\_format},  {'name': 'Sales Volume', 'id'; 'Sales Volume'}]  d\_table = DataTable(columns = d\_columns, data = df.to\_dict('records'),  cell\_selectable = False, sort\_action='native', filter\_action='native'  page\_current = 0, page\_size = 2, page\_action = 'native'  style\_cell = ({'textAlign': 'left'}),  style\_cell\_conditional=[{'if': {'column\_id': 'Sales Volume'}, 'textAlign': 'center'}])  style\_header = {'background-color': 'black', 'color': 'white'}  html.Div(d\_table) | | | | | | | | | | 2 d.p  #name is shown to user  #pandas to\_dict records: list like  sort and filter can be set to custom  pagination  same as CSS  conditional styling  style\_header\_conditional also can |
| set cell\_selectable = True  @app.callback(  Output('text', 'children'),  Input('my\_dt', 'selected\_cells'))  def print\_it(input):  return str(input) | | | send input from selected cells  #also can set row\_selectable to single or multi  #Input then becomes Input('my\_dt', 'selected\_rows')  #for column, set column\_selectable to 'single' or 'multi'  #add 'selectable': True to col definition in d\_columns  #Input('my\_dt', 'selected\_columns') | | | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Shiny | ui is an R obj that defines:  - input elements that user iteracts with  - output elements that app will render  - layout of these 2 elems  - input and output elems have an id | | serveris an R obj that defines the logic of the app:  - what to do when a particular input elem is changed  - server auto detects and reacts to changes in input elems  - input elem are referenced here using input$\_\_\_ notation within reactive code sections  - output elems are referenced using output$\_\_\_ notation | | | |
|  |  | | | | | |
|  | sidebarPanel(  sliderInput(),  selectInput("color",  choices = c("Red" = "red", "Brown" = "brown", "Violet" = "violet"),  label = "Choose a color")  ),  mainPanel(  plotOutput("distPlot"),  textOutput("color\_choice")  ) | | | | server <- function(input, output) {  output$distPlot <- renderPlot({  # generate bins based on input$bins from ui.R  x <- faithful[, 2]  bins <- seq(min(x), max(x), length.out = input$bins + 1)  # draw the histogram with the specified number of bins  hist(x, breaks = bins, col = *input$color*, border = 'white',  xlab = 'Waiting time to next eruption (in mins)',  main = 'Histogram of waiting times')  })  output$color\_choice <- renderText((  paste("The color that you chose was ", input$color)  ))  } | |
| docker | | docker run --name shiny-01 -d -p3838:3838 rocker/shiny  docker cp <folder> <container>:/srv/shiny-server/  docker cp <folder>/ <container>:/srv/shiny-server/  docker exec <bash command>  open terminal in docker app, cd to /var/log/shiny-server | | #want to add folder w own app into container running shiny server  #copy folder on local machine w files into container folder  #copy only files into container folder  #view log files to troubleshoot | | |
| HTML | | Block elements (row by row)  <p>: paragraph  <ol>: ordered list  navigation menus, footers, div elements | | | Inline elements (side by side)  <a>: anchor elements  <em>: italicise text  <strong>: bold text | |
| Can add elements attributes, won't appear in rendered output | | | <p class = "..."> ... </p> (common attributes are class, id) | |
|  | | <!DOCTYPE html>  <html lang="en-US">  <head>  <meta charset="utf-8" />  <title>My test page</title>  </head>  <body>  <p>This is my page</p>  </body>  </html> | | | - DOCTYPE: historical artifact  - html: root elements. Wraps all content on page  - head: content here does not appear on page. Used to declare styles or Javaascript functions  - title: appears on browser tab. Also appears in bookmarks page  - body: content appearing on page | |
| Running a web-server | | 1a) activate python virtual environment 1b) python -m http.server  2) docker run --name test-nginx -p8000:80 --rm -v <path-to-html-pages>:/usr/share/nginx/html -d nginx | | | --rm: remove container after stopping | |
| CSS | | h1 {color: red; font-size: 5em;}  h1.special {}  .special {}  #unique {}  A, B {}  A B | | | | All h1 tag will have this CSS  H1 tag with class special  All tags with class special  All tags with id = unique  Apply rule to A or B tag  Apply rule to tag B nested within A |
|  | | To incorporate CSS:  1) Create css file and in HTML file <head> do <link rel="stylesheet" href="styles.css" />  2) Include all rules in document header, under <style> elements  3) Specify style for specific elem, <h1 style="color: blue; border: 1px black;"> Hello World! </h1> | | | | |
|  | |  | | |  | |
|  | |  | | |  | |
|  | |  | | |  | |
|  | |  | | |  | |
|  | |  | | |  | |
|  | |  | | |  | |