Linux

WILDCARDS: *, ?, [char1char2], [char1-char2], {seq1,seq2}
Example: `ls desk/[Pp]aper[0-9]*.{txt,pdf}`

```
chown user:group path/to/file_or_directory
chown -R user path/to/directory

chgrp group path/to/file_or_directory
chgrp -R group path/to/directory
```

```
       u   r       rw
chmod  g + w   or  rx
       o - x       …
```

my_files=(/path/to/files/*.csv)

```
for item in ARRAY_ITEMS; do
   command 1
   command 2
   ..
done         e.g. ${sw_crew[@]}
```

```
for (( I=0; I<=${#sw_crew[@]}; I+=1 )); do
   echo "Member: ${crew_size[I]}"
done
```

```
if [[ X BIN_OP Y ]]; then
     set of commands
elif [[ X BIN_OP Y ]]; then
     other set of commands
fi
```

```
if [[ UNI_OP X ]]; then
     commands
fi
```

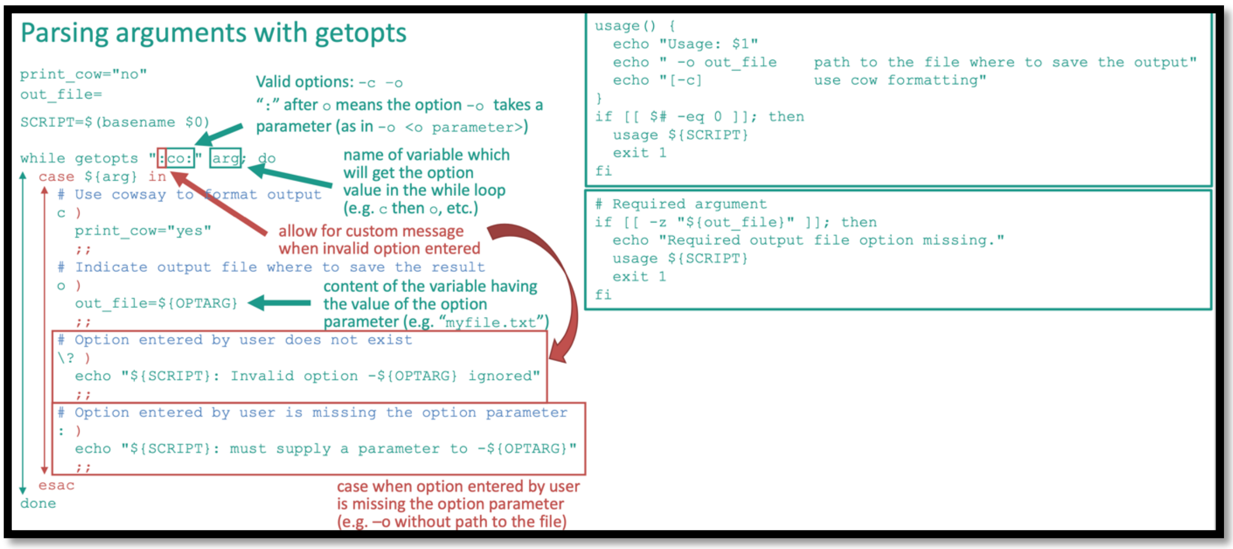
```
crew_size=${#sw_crew[@]}
K=0
while [[ $K -lt $crew_size ]]; do
  echo "Crew member: ${sw_crew[K]}"
  K=$(( K + 1 ))
done
```

```
read yes_or_no # yes_or_no now has the answer of the user

case $yes_or_no in
  [yY][Ee][Ss] )
    echo "You agreed"
    ;;
  [nN][Oo] )
    echo "You did not agree"
    ;;
  *)
    echo "Invalid input. Please answer yes or no."
    ;;
esac
```

```
my_function() {
  commands using or not $1, $2, etc.
}
```

Variable	description
$0	The command the script was called with
$1	The first argument of the script
$2	The second argument of the script
…	
$#	The number of arguments

`echo ${my_var[@]}` to return the whole array
`echo ${my_var[*]}` to return a single string joining all items
`echo ${my_var[K]}` to return the Kth item (starting from 0!)
`echo ${#my_var[@]}` to return the total number of items



Parsing arguments with getopts

```
print_cow="no"
out_file=

SCRIPT=$(basename $0)

while getopts ":co:" arg; do
  case ${arg} in
    # Use cowsay to format output
    c )
      print_cow="yes"
      ;;
    # Indicate output file where to save the result
    o )
      out_file=${OPTARG}
      ;;
    # Option entered by user does not exist
    \? )
      echo "${SCRIPT}: Invalid option -${OPTARG} ignored"
      ;;
    # Option entered by user is missing the option parameter
    : )
      echo "${SCRIPT}: must supply a parameter to -${OPTARG}"
      ;;
  esac
done
```

Valid options: `-c -o`
":" after `o` means the option `-o` takes a parameter (as in `-o <o parameter>`)

name of variable which will get the option value in the while loop (e.g. `c` then `o`, etc.)

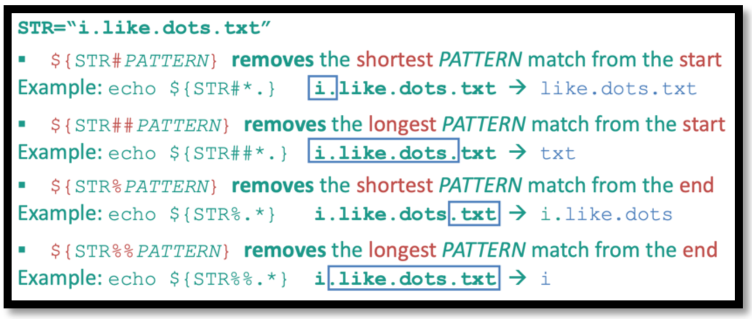
allow for custom message when invalid option entered

content of the variable having the value of the option parameter (e.g. "`myfile.txt`")

case when option entered by user is missing the option parameter (e.g. –o without path to the file)

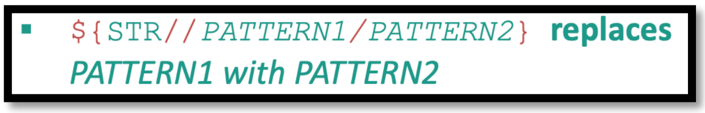
```
usage() {
  echo "Usage: $1"
  echo " -o out_file     path to the file where to save the output"
  echo "[-c]             use cow formatting"
}
if [[ $# -eq 0 ]]; then
  usage ${SCRIPT}
  exit 1
fi
```

```
# Required argument
if [[ -z "${out_file}" ]]; then
  echo "Required output file option missing."
  usage ${SCRIPT}
  exit 1
fi
```



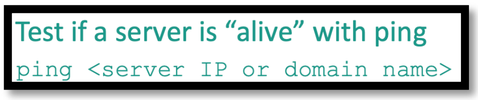
`STR="i.like.dots.txt"`

- `${STR#PATTERN}` **removes** the shortest *PATTERN* match from the start
  Example: `echo ${STR#*.}` `i.like.dots.txt` → `like.dots.txt`
- `${STR##PATTERN}` **removes** the longest *PATTERN* match from the start
  Example: `echo ${STR##*.}` `i.like.dots.txt` → `txt`
- `${STR%PATTERN}` **removes** the shortest *PATTERN* match from the end
  Example: `echo ${STR%.*}` `i.like.dots.txt` → `i.like.dots`
- `${STR%%PATTERN}` **removes** the longest *PATTERN* match from the end
  Example: `echo ${STR%%.*}` `i.like.dots.txt` → `i`



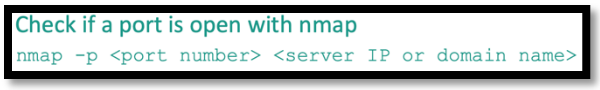
- `${STR//PATTERN1/PATTERN2}` **replaces** *PATTERN1 with PATTERN2*

IP + Port = Socket (52.212.137.39:8080)



Test if a server is "alive" with ping

```
ping <server IP or domain name>
```



Check if a port is open with nmap

```
nmap -p <port number> <server IP or domain name>
```

ssh-keygen -t rsa -b 4096
ssh-copy-id username@remote_host

Git

- man git-command (e.g. man git-commit)
- git init
- git add .
- git commit -m “message”
- commit includes checksum, previous checksum, message, author, and date
- git status
- git log (--oneline OR –summary)
- git show
- git diff OldCommit NewCommit
- git difftool OldCommit NewCommit
- gitk
- git config --global user.name “NAME”
- git config --global user.email “email”

Branches

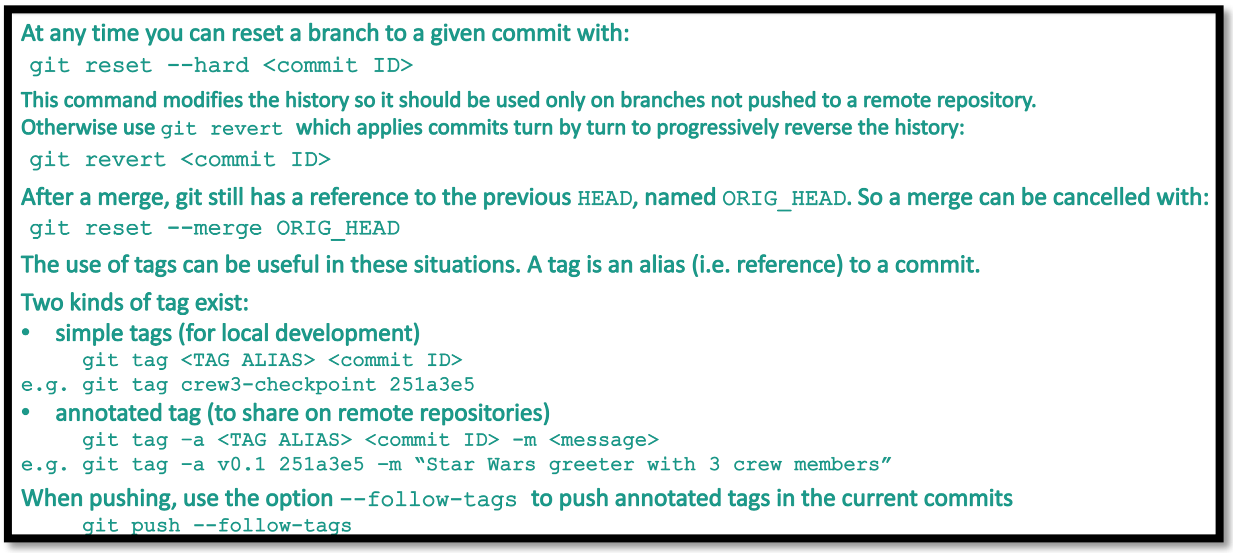
- git branch topic
- git checkout topic
- git branch topic <COMMIT-id> → creates the topic branch at COMMIT-id
- branch is a pointer to a commit
- HEAD is a pointer to the active branch
- git branch -v → lists all the branches
- git branch -d topic → deletes the branch
- git branch -M main → renames the branch

Remote

- git remote add origin git@github.com:<user>/<repo>.git
- git clone
- git push -u origin main
- git pull origin main (same as: git fetch origin main + git merge origin/main)
- git branch -avv (shows all local and remote branches)
- git branch -u origin/main (restores the remote tracking in the now github repo)

Merge

- If main branch is paused:
  - git checkout main
  - git merge topic (we can use --no-ff here to keep the history)
- If 3 way:
  - git checkout topic
  - git rebase main
  - git checkout main
  - git merge topic --no-ff
- If 3 way:
  - git checkout main
  - git cherry-pick <commit-id> (or <commit-id1>~..<commit-id2>)
  - first commit-id is ignored.
  - ~ means one before



At any time you can reset a branch to a given commit with:

```
git reset --hard <commit ID>
```

This command modifies the history so it should be used only on branches not pushed to a remote repository.
Otherwise use `git revert` which applies commits turn by turn to progressively reverse the history:

```
git revert <commit ID>
```

After a merge, git still has a reference to the previous `HEAD`, named `ORIG_HEAD`. So a merge can be cancelled with:

```
git reset --merge ORIG_HEAD
```

The use of tags can be useful in these situations. A tag is an alias (i.e. reference) to a commit.

Two kinds of tag exist:

- simple tags (for local development)

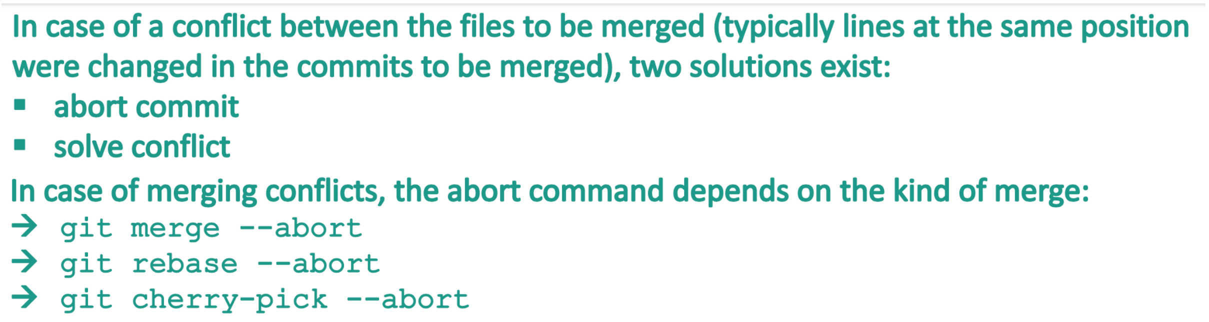
```
     git tag <TAG ALIAS> <commit ID>
e.g. git tag crew3-checkpoint 251a3e5
```

- annotated tag (to share on remote repositories)

```
     git tag -a <TAG ALIAS> <commit ID> -m <message>
e.g. git tag -a v0.1 251a3e5 -m "Star Wars greeter with 3 crew members"
```

When pushing, use the option `--follow-tags` to push annotated tags in the current commits

```
     git push --follow-tags
```



In case of a conflict between the files to be merged (typically lines at the same position were changed in the commits to be merged), two solutions exist:

- abort commit
- solve conflict

In case of merging conflicts, the abort command depends on the kind of merge:

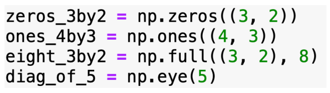
```
→ git merge --abort
→ git rebase --abort
→ git cherry-pick --abort
```

If we want to solve the conflict, we can use “git mergetool”

Python

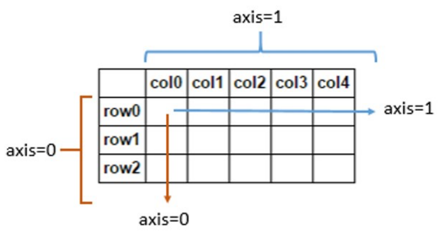
```
def comment_grade(grade: int, mode: str = 'normal') -> str :
  “””
     Provide a feedback according to the grade value

      Parameters
      ----------
      grade
         The grade obtained by the student (out of 10)
      mode
         The feedback mode, either "normal" (default) or "positive_reinforcement"

      Returns
      -------
      comment
         The grade feedback

      Examples
      -------------
      >>>  comment_grade(6)
      ‘Grade high enough’
  “””
  If1 grade >= 0 and grade < 5:
      return('Grade too low')
  elif1 grade > 5 and grade <= 10:
      if2 mode == 'normal':
         return('Grade high enough')
      elif2 mode == 'positive_reinforcement':
         return('Well done, keep going!')
      else2:
         raise ValueError('The mode should be "normal" or "positive_reinforcement"')
  else1:
      assert (grade < 0 or grade > 10), 'INTERNAL BUG: grade is not less than 0 or greater than 10'
      raise ValueError('EXTERNAL ERROR: The grade entered should be between 0 and 10')
```

- os.getcwd()
- sys.path
- os.__file__
- if __name__==”__main__”: (if executed from *command line*)
- import importlib; importlib.reload(numpy)
- def test_grade():
  ```
          assert comment_grade(0, “normal”) == ”Grade too low”
  ```
- pytest --doctest-modules file.py
- class Student:
  ```
          def __init__(self, first_name, last_name, grade=None):
                  self.first_name=first_name
                  self.last_name=last_name
                  self.grade=grade
          def get_name():
                  return self.first_name + “ “ + self.last_name
  ```
- docstring for the class has Attributes and Methods
- student1 = Student(Ali, Saadat, 20)

- 

```
zeros_3by2 = np.zeros((3, 2))
ones_4by3 = np.ones((4, 3))
eight_3by2 = np.full((3, 2), 8)
diag_of_5 = np.eye(5)
```

- Reading files:
    - home_path=os.path.expanduser('~')
    - file_path=os.path.join(home_path, "Documents", "data", "file.csv")
    - with open(file_path, 'r') as file_handler:

      file_content=file_handler.read()
- all_files=sort(glob.glob("~/data/*.csv"))
- patient_files=[f for f in all_files if not "small" in f]
- np.loadtext(fname="/path/to/file", delimeter=',')
- 

- df = pd.readcsv("/path/to/file")
    - df.describe()
    - df.head(5)
    - df.shape
    - df.dtypes
    - df.isna().sum()
    - df[df['year]>=2000]
    - df.dropna()
    - df['country'].value_counts()
    - top10 = list(df['country'].value_counts().nlargest(10).index)
      top10_mask=df['country'].isin(top10)
      df[top10_mask]
    - df['country'].value_counts().nlargest(10).plot(kind='bar')
- sns
    - sns.boxplot(df['year'])
    - sns.histplot(df['year'])
    - sns.kdeplot(df['year'], df['price'])
    - sns.scatterplot(x='year', y='price', data=df)
    - sns.countplot(y='country', data=df)
    - sns.violinplot(x="price", y="country", data=df)
    - with sns.plotting_context("notebook", font_scale=1.2):

      g = sns.catplot(x="model", y="val", hue="stage", col="scorer",

      data=train_test_results_df, kind="bar", sharey=False)

ML

Train_test_split

-
  ```
  from sklearn.model_selection import train_test_split
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
  random_state=42)
  ```

LinearRegression, MSE, R2

-
  ```
  from sklearn.linear_model import LinearRegression
  from sklearn.metrics import mean_squared_error, r2_score
  lm = LinearRegression()
  lm.fit(X_train, y_train)
  y_pred = lm.predict(X_train)
  R2_train = r2_score(y_train, y_pred)
  MSE_train = mean_squared_error(y_train, y_pred)
  ```

Pipeline

-
  ```
  from sklearn.preprocessing import PolynomialFeatures
  from sklearn.pipeline import Pipeline
  lm_deg5 = Pipeline([('poly_transformer', PolynomialFeatures(degree=5)),
                  ('lm', LinearRegression())])
  lm_deg5.fit(X_train, y_train)
  lm_deg5['lm'].coef_
  y_pred = lm_deg5.predict(X_train)
  ```

CrossValidation

-
  ```
  from sklearn.model_selection import KFold
  ml_models = {'lm': LinearRegression(),
           'lm_deg2': Pipeline([('poly_transformer', PolynomialFeatures(degree=2)),
                       ('lm', LinearRegression())]),
           'lm_deg5': Pipeline([('poly_transformer', PolynomialFeatures(degree=5)),
                       ('lm', LinearRegression())])}
  kf_results = []

  kfs = KFold(n_splits=10, shuffle=True, random_state=42)
  for i_f, (ix_train, ix_test) in enumerate(kfs.split(X_train)):
     # Loop over models
     for mod_name, mod in ml_models.items():
        # Define training and testing folds
        X_training_folds = X_train.iloc[ix_train]
        y_training_folds = y_train.iloc[ix_train]
        X_test_fold = X_train.iloc[ix_test]
        y_test_fold = y_train.iloc[ix_test]
        # Fit the model on the training folds
         mod.fit(X_training_folds, y_training_folds)
        # Test on both the training and testing folds to check for over-/under-fitting
        y_pred_train = mod.predict(X_training_folds)
        y_pred_test = mod.predict(X_test_fold)
        # R2
        kf_results.append({'model': mod_name, 'fold': i_f, 'stage': 'train', 'scorer': 'r2',
  ```

CrossValidation automated

-
  ```
  from sklearn.model_selection import cross_val_score
  from sklearn.metrics import fbeta_score, make_scorer
  mse_scorer = make_scorer(mean_squared_error, greater_is_better=False)
  cv_test_scores = {}
  for mod_name in ml_models.keys():
     cv_test_scores[mod_name] = cross_val_score(ml_models[mod_name], X_train, y_train,
                                  cv=kfs, scoring=mse_scorer, n_jobs=-1)
  cv_test_scores_df = pd.DataFrame(cv_test_scores)
  ```

CrossValidation aturomated2

-
  ```
  from sklearn.model_selection import cross_validate
  cv_scores = {}
  for mod_name in ml_models.keys():
     cv_scores[mod_name] = cross_validate(ml_models[mod_name], X_train, y_train, cv=kfs,
                              scoring=['r2', 'neg_mean_squared_error'],
                              return_train_score=True, n_jobs=-1)

  def crossval_to_df(cv_dict):
     crossval_results = []
     for model in cv_dict.keys():
        for scorer in cv_dict[model].keys():
           if scorer.startswith('train_'):
              score = scorer.replace('train_', '')
              for i_val, val in enumerate(cv_dict[model][scorer]):
                 crossval_results.append({'model': model, 'fold': i_val, 'stage': 'train',
                              'scorer': score, 'val': val})
           elif scorer.startswith('test_'):
              score = scorer.replace('test_', '')
              for i_val, val in enumerate(cv_dict[model][scorer]):
                 crossval_results.append({'model': model, 'fold': i_val, 'stage': 'test',
                              'scorer': score, 'val': val})
     return pd.DataFrame(crossval_results)

  crossval_df = crossval_to_df(cv_scores)
  ```

Transform Label and Features

-
  ```
  from sklearn.preprocessing import LabelEncoder, OrdinalEncoder, OneHotEncoder,
  StandardScaler
  from sklearn.compose import make_column_transformer

  y_num = LabelEncoder().fit_transform(y_cat)
  cols_ordinal = ["smoking_status"]
  cols_non_ordinal = ["gender", "ever_married", "work_type", "Residence_type"]
  cols_continuous = ["age", "avg_glucose_level", "bmi"]
  preprocessor = make_column_transformer(
     (OrdinalEncoder(categories=[['never smoked', 'formerly smoked', 'smokes']]), cols_ordinal),
     (OneHotEncoder(drop='if_binary', sparse=False), cols_non_ordinal),
     (StandardScaler(), cols_continuous),
     remainder='passthrough',
     verbose_feature_names_out=False
  )
  X_final = pd.DataFrame(data=preprocessor.fit_transform(X),
                 columns=preprocessor.get_feature_names_out())
  ```

Logistic Regression

-
  ```
  from sklearn.linear_model import LogisticRegression
  lr = LogisticRegression(penalty='none', class_weight='balanced', max_iter=1000)
  lr.fit(X_train, y_train)
  ```

Confusion Matrix

-
  ```
  from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
  y_pred_train = lr.predict(X_train)
  accuracy_score(y_train, y_pred_train)
  confusion_matrix(y_train, y_pred_train)
  ConfusionMatrixDisplay.from_estimator(lr, X_train, y_train,
  display_labels=label_encoder.classes_,
                                normalize='true')
  lr_coefs_df = pd.DataFrame({'coefs': np.std(X_train, 0)*lr.coef_[0]},
                        index=X.columns)
  plt.figure(figsize=(6,6))
  sns.barplot(x='coefs', y=lr_coefs_df.index, data=lr_coefs_df)
  ```

PCA

-
  ```
  from sklearn.datasets import load_digits
  from sklearn.decomposition import PCA
  digits = load_digits()
  pca = PCA(n_components=2)
  projected_digits = pca.fit_transform(digits.data)

  pca = PCA().fit(digits.data)
  plt.plot(np.cumsum(pca.explained_variance_ratio_))
  ```

Kmeans

-
  ```
  from sklearn.cluster import KMeans
  from sklearn.preprocessing import scale
  X = scale(digits.data)
  y = digits.target
  X_pca = PCA(n_components=2).fit_transform(X)
  kmeans = KMeans(n_clusters=n_digits)
  kmeans.fit(X_pca)
  ```

silhouette

-
  ```
  from sklearn.metrics import silhouette_samples, silhouette_score
  clusterer = KMeans(n_clusters=n_clusters, random_state=10)
  cluster_labels = clusterer.fit_predict(X_pca)
  silhouette_avg = silhouette_score(X_pca, cluster_labels)
  sample_silhouette_values = silhouette_samples(X_pca, cluster_labels)
  ```