Setting development environment (Unix)

======================================

Install all required development packages: GNU make, g++, ...

Build:

```

cd gdal

./configure [options]

make -j8 -s

cd apps; make -s test\_ogrsf; cd ..

```

Run command line utilities (without installing):

```

. scripts/setdevenv.sh

gdalinfo --version

```

Run autotest suite:

```

cd ../autotest

pip install -r requirements.txt

pytest

```

Git workflows with GDAL

=======================

This is not a git tutorial or reference manual by any means. This just collects

a few best practice for git usage for GDAL development.

Commit message

--------------

Indicate a component name (eg a driver name), a short description and when

relevant, a reference to a issue (with 'fixes #' if it actually fixes it)

```

COMPONENT\_NAME: fix bla bla (fixes #1234)

Details here...

```

Initiate your work repository

-----------------------------

Fork OSGeo/gdal from github UI, and then

```

git clone https://github.com/OSGeo/gdal

cd gdal

git remote add my\_user\_name https://github.com/my\_user\_name/gdal.git

```

Updating your local master against upstream master

--------------------------------------------------

```

git checkout master

git fetch origin

# Be careful: this will loose all local changes you might have done now

git reset --hard origin/master

```

Working with a feature branch

-----------------------------

```

git checkout master

(potentially update your local master against upstream, as described above)

git checkout -b my\_new\_feature\_branch

# do work. For example:

git add my\_new\_file

git add my\_modifid\_message

git rm old\_file

git commit -a

# you may need to resynchronize against master if you need some bugfix

# or new capability that has been added since you created your branch

git fetch origin

git rebase origin/master

# At end of your work, make sure history is reasonable by folding non

# significant commits into a consistent set

git rebase -i master (use 'fixup' for example to merge several commits together,

and 'reword' to modify commit messages)

# or alternatively, in case there is a big number of commits and marking

# all them as 'fixup' is tedious

git fetch origin

git rebase origin/master

git reset --soft origin/master

git commit -a -m "Put here the synthetic commit message"

# push your branch

git push my\_user\_name my\_new\_feature\_branch

From GitHub UI, issue a pull request

```

If the pull request discussion or Travis-CI/AppVeyor checks require changes,

commit locally and push. To get a reasonable history, you may need to

```git rebase -i master```, in which case you will have to force-push your

branch with ```git push -f my\_user\_name my\_new\_feature\_branch```

Backporting bugfixes from master to a stable branch

---------------------------------------------------

```

git checkout master

With git log, identify the sha1sum of the commit you want to backport

git checkout 2.2 (if you want to backport to 2.2)

git pull origin 2.2

(git checkout -b branch\_name: if you intend to submit the backport as a pull request)

git cherry-pick the\_sha1\_sum

git push ...

```

If changes are needed, do them and ```git commit -a --amend```

Things you should NOT do

------------------------

(For anyone with push rights to github.com/OSGeo/gdal) Never modify a commit or

the history of anything that has been

committed to https://github.com/OSGeo/gdal