GIT cheatsheet

Basics and SVN equivalents

 # Fetch copy of repository (svn:checkout)
```
git clone --branch branch_name repository_url
```

 # CHECKOUT (switch to branch)
```
git checkout branch_name
```

 # PULL (svn: update)
```
git pull origin branch_name
```

 # STATUS (svn: status)
```
git status
```

 # ADD untracked files (svn: add)
```
git add [directory or file]
git add *
```

 # COMMIT to local (no svn equiv)
```
git commit -am "[comment]"
```

 # PUSH to remote (svn:commit)
```
git push origin branch_name
```

Roll back

 # before commit - undo local changes
```
git reset --hard
```

 # after commit - undo changes of a previous commit
```
git revert [commit's hash number]
```

Current hash

 # get hash of latest commit in current branch
```
git rev-parse HEAD
```

Commit crediting another developer

 # use author option

```
git commit --author="username <username@userid.no-reply.drupal.org>" \
-am "Issue #issue_number by username: Fixed whatever."
```

Branches

 # create branch
```
git checkout -b branch_name
```

 # tell remote
```
git push origin branch_name
```

 # show local branches
```
git branch -v
```

 # show local and remote branches
```
git branch -a
```

 # refresh local list of remote branches
```
git remote prune origin
```

Make branch up-to-date versus trunk (~ the develop branch)

You work in a feature branch, originally created as a copy of the develop branch. Meanwhile, other feature branches have been merged into develop: your branch is *behind* develop.

 # switch to your feature branch
```
git checkout feature_branch
```

 # apply recent changes in develop upon your feature branch
```
git fetch; git pull origin develop
```

Do this frequently - otherwise merges will require complex and unsafe manual work.

Merge conflict... favour changes of current or other branch

 # favour changes of the other
```
git pull origin branch_name -X theirs
```

 # favour changes of current (rarely relevant)
```
git pull origin branch_name -X ours
```

Is feature branch up-to-date with develop?

```
git checkout feature_branch
git fetch; git pull origin develop –no-commit
```

Merge feature branch into develop

 # refresh local feature branch

```
git checkout feature_branch
git fetch; git pull origin feature_branch
```

 # refresh local develop

```
git checkout develop
git pull origin develop
```

 # merge

```
git merge feature_branch
git push origin develop
```

List branches merged into develop

 # must be 'on' the develop branch

```
git checkout develop
git branch -a --merged develop
```

Delete branch

 # delete local

```
git branch -d branch_name
```

 # delete remote

```
git push origin :branch_name
```

Clone a single file

```
git archive --remote=repository_url refs/heads/branch_name \
path/to/file |tar xf -
```

Tags

 # create tag
```
git tag tag_name
```

 # tell remote
```
git push --tags
```

 # checkout tag
```
git checkout tag_name
```

 # list local tags
```
git tag
```

 # list remote tags
```
git ls-remote --tags origin
```

(don't) Remove tag

You must be *absolutely* sure that no one/nothing expects (depends on) the tag!

 # remove local tag
```
git tag -d tag_name
```

 # remove remote tag
```
git push origin :refs/tags/tag_name
```

 # remove remote tag in drupal.org's (weird) git
```
git push origin :tags/tag_name
```

Create git patch

 # fetch basis branch
```
git clone --branch basis_branch_name repository_url
```

 # create new branch
```
git checkout -b fixing_branch_name
```

Work...

 # create patch against the basis branch
issue number and comment number might be relevant in your context
```
git diff basis_branch_name > project_name-short-description-issueno-commentno.patch
```

Create non-git patch (single file)

 # create patch against copy of original
```
diff -Nau ../compare-with/copy-of-original.file subject.file > the.patch
```

Correct file handles within the patch; won't work when applying. Patch says:

```
--- ../compare-with/copy-of-original.file YYYY-MM-DD
+++ subject.file YYYY-MM-DD
```

But should be:
```
--- a/subject.file YYYY-MM-DD
+++ b/subject.file YYYY-MM-DD
```

Apply patch

 # place yourself in the root dir of the project
```
cd whatever/project_name
```

 # place the patch in the root dir of the project
```
wget --no-check-certificate https://url/patch_name.patch
```

 # apply the patch via git
if the project is checked out from git
```
git apply -v patch_name.patch
```

 # apply without git
project not checked out from git
```
patch -p1 < patch_name.patch
```

Remote (repo) url

 # show
```
git remote -v
```

 # change
```
git remote set-url origin git@[host]:[group]/[project].git
```

Migrate repo

 # make a bare clone of the old server repository to a local directory
```
git clone --bare git@old-host:group/project.git
```

 # push mirror to new server repository
```
cd project.git; git push --mirror git@new-host:group/project.git
```

Passive secondary mirror

Add a secondary mirror, to reflect the (primary) origin.

 # add remote mirror
```
git remote add --mirror=fetch mirrorname git@mirror-host:group/project.git
```

 # reflect origin to mirror
```
git fetch origin
git push mirrorname --all; git push mirrorname --tags
```

Shrink repo / remove unwanted files from history

Use git filter-repo or BFG Repo Cleaner (this example).

Beware that all involved commits will get new hashes.

 # make a bare clone

```
git clone --bare git@old-host:group/project.git
```

 # use BFG to mark 'target.file' as garbage in all history of all branches

```
java -jar bfg-N.N.N.jar --delete-files target.file project.git
```

 # run garbage collection

```
cd project.git
git reflog expire --expire=now --all && git gc --prune=now --aggressive
```

 # update remote

```
git push origin --all --force
git push origin --tags --force
```

Make git ignore file mode

So you chmod'ed a checked out project, and suddenly git status reports all files changed...

Before chmod'ing, do (either or both):

 # ignore in current project, in the future

```
git config core.filemode false
```

 # ignore for all projects, in the future

```
git config --global core.filemode false
```

Apply .gitignore changes

Changing a .gitignore does not automagically make dirs|files included|excluded *locally*.

 # reset local state

```
git rm -r --cached .; git add .
```

.gitignore dir except for some file(s)

>> An optional prefix "!" which negates the pattern; any matching file excluded by a previous pattern will become included again.
It is not possible to re-include a file *if a parent directory of that file is excluded*. <<

/dir/
= excludes dir and all children of dir

/dir/*
!/dir/some.file
= excludes dir except for some.file

/dir/*
!/dir/sub-dir
/dir/sub-dir/*
!/dir/sub-dir/some.file
= excludes dir except for sub-dir/some.file

Make git ignore a sub dir's foreign .git - prevent sub-moduling

Git sub-moduling is a pain, to be avoided. Instead...

Example: A module includes a library, which belongs to another git repo.
The module's releases must include the library's source.
But during development, the library should (primarily) reflect it's own git repo.

1. Add a .gitignore file to the module, containing (path to library's .git):
   vendor/*/*/.git
2. git commit
3. Add the library's source
4. git add vendor/*
5. git commit

PHP Composer and git - almost incompatible

Git *does* actually work in combination with composer.
But console messages will be misleading.
You'll be working in the dark, with a high risk of making mistakes.

1. Composer adds itself as another - and masking - origin in a project's .git.
   You could do 'git remote rm composer', but no permanent fix.
2. 'git status' doesn't compare against origin, at least doesn't tell in console.
   Composer effectively only works with https as *fetch* url - composer fetching via ssh takes several minutes per project.
3. Composer always reads dependencies from the git *master* branch.
   Even when you've told composer that it should use a develop branch.
   And there's no fix; merge dependency changes into master, or go to ####.

Solution to 1 & 2

Keep a copy of your project's, which you yourself have cloned from git.

Before 'composer install'/'composer update'

Move project cloned directly from git - away from the 'vendor' dir.

After 'composer update'

Delete project cloned by composer. Move project cloned directly, back in place.