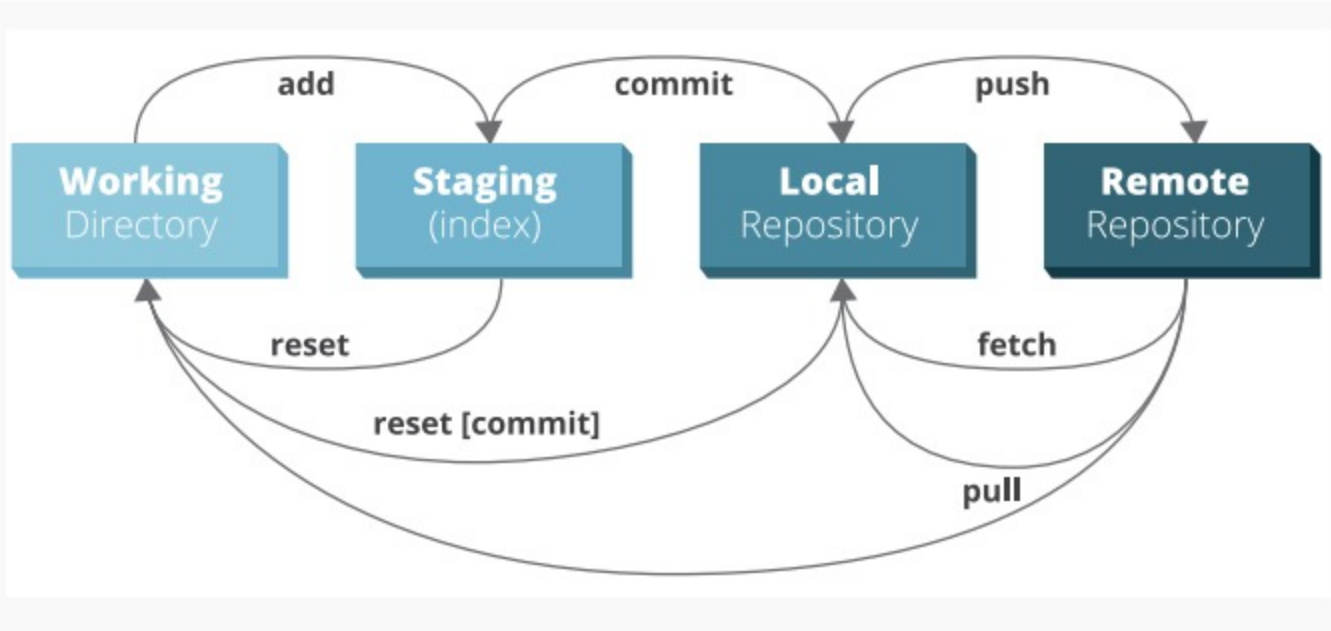
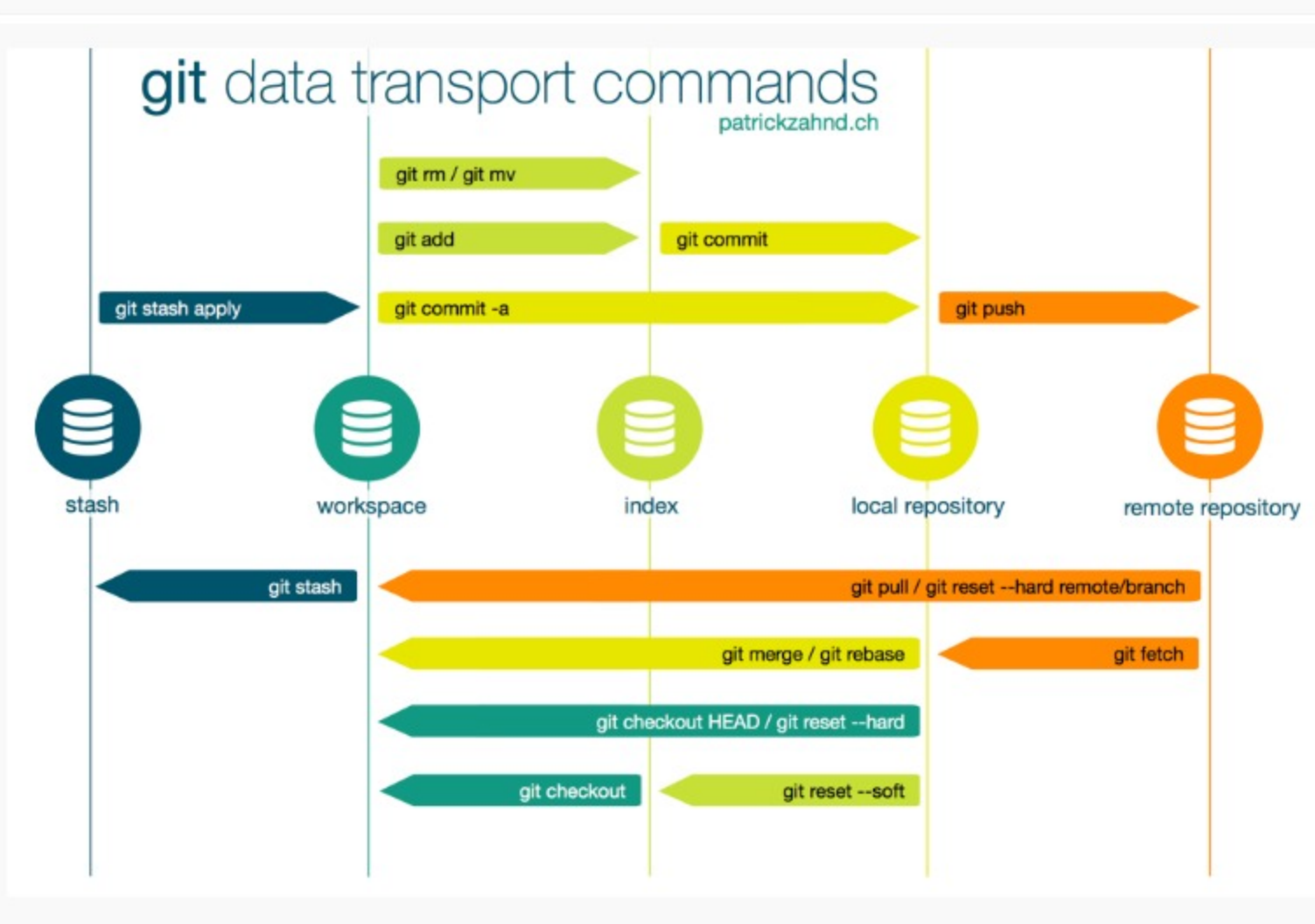
Git Cheat Sheet: Useful Commands, Tips and Tricks

The Git Workflow



Git Workflow

Taken from http://files.zeroturnaround.com/pdf/zt_git_cheat_sheet.pdf



Git in Time Workflow

Taken from patrickzahnd.ch

Configuration

```
# show current values for all global configuration parameters
git config --list --global

# set a username globally
git config --global user.name "username"`

# set an email address globally
git config --global user.email "email@provider.com"

# always --prune for git fetch and git pull
git config --global fetch.prune true

# remove the previously set username globally
git config --global --unset user.name

# color the git console
git config color.ui true

# set the tool used by git for diffing globally
git config --global diff.tool mytool

# set the tool used by git for merging globally
git config --global merge.tool mytool

# git config --global --unset myparameter
remove the previously set configuration value globally

# allows populating the working directory sparsely, that is, cloning only certain directories from a repository
git config core.sparseCheckout true

# instruct git to retrieve only some directory in addition to those listed in `.git/info/sparse-checkout
echo "some/directory/inside/the/repository" >> .git/info/sparse-checkout

# define which whitespace problems git should recognize(any whitespace at the end of a line, mixed spaces or tabs)
git config --global core.whitespace trailing-space,space-before-tab

# tells Git to detect renames. If set to any boolean value, it will enable basic rename detection. If set to "copies" or "copy", it will detect copies, as well.
git config --global diff.renames copies

# if set, git diff uses a prefix pair that is different from the standard "a/" and "b/" depending on what is being compared.
git config --global diff.mnemonicprefix true

# always show a diffstat at the end of a merge
git config --global merge.stat true

# no CRLF to LF output conversion will be performed
git config --global core.autocrlf input

# whenever pushing, also push local tags
git config --global push.followTags true

# show also individual files in untracked directories in status queries
git config --global status.showUntrackedFiles all

# always decorate the output of git log
git config --global log.decorate full

# the git stash show command without an option will show the stash in patch form
git config --global stash.showPatch true

# always set the upstream branch of the current branch as the branch to be pushed to when no refspec is given
git config --global push.default tracking
```

Git .ignore Syntax

`file`	match a particular file
`.file`	match a hidden file
`directory/`	match a directory
`directory/directory/`	match a subdirectory
`directory/directory/*.extension`	match all files with a certain extension in a subdirectory
`directory/directory/**/*.extension`	recursively match all files with a certain extension in a subdirectory
`/*`	match everything
`!file`	do not match file

Git Workflow

Initialize and Clone

```
# initialize a git repository in the current working directory
git init

# clone a remote repository over https
git clone https://remote.com/repo.git

# clone a remote repository over ssh
git clone ssh://git@remote.com:/repo.git

# recursively clone a repository over https
git clone --recusive https://remote.com/repo.git

# recursively clone a repository over ssh
git clone --recursive ssh://git@remote.com:/repo.git
```

Track, Add and Commit

```
# tell git to start tracking a file or add its current state to the index
git add file

# tell git to add everything which is untracked or has been changed to the index
git add .

# commit to local history with a given message
git commit -m "message"

# add all changes to already tracked files and commit with a given message, non-tracked files are excluded
git commit -am "message"

# modify the last commit including both new modifications and given message
git commit --amend -m "message"

# perform a commit with an empty message
git commit --allow-empty-message -m
```

Status and Diagnostics

```
# show the commit at the head of the branch currently checked out
git show HEAD

# shows the commit whose object ID matches mycommit
git show mycommit

# shows the status of the local git repository
git status

# shows a short version of the status
git status -s
```

Checking Out

```
# replace filename with the latest version from the current branch
git checkout -- filename

# in case fileorbranch is a file, replace fileorbranch with the latest version of the file on the current branch.
# in case fileorbranch is a branch, replace the working tree with the head of said branch.
git checkout fileorbranch

# replace the current working tree with commit 05c5fa
git checkout 05c5fa

# replace the current working tree with the head of the master branch
git checkout master
```

Working with Remotes

```
# show the remote branches and their associated urls
git remote -v

# adds an https url as remote branch under the name origin
git remote add -f origin https://remote.com/repo.git

# adds an ssh url as remote branch under the name origin
git remote add -f origin ssh://git@remote.com:/repo.git

# remove the remote with ID origin
git remote remove -f origin

# set an https url for the remote with ID origin
git remote set-url origin https://remote.com/repo.git

# set an ssh url for the remote with ID origin
git remote set-url origin ssh://git@remote.com:/repo.git

# clean up remote non-existent branches
git remote prune origin

# set the upstream branch, to which changes will be pushed, to origin/master
git branch --set-upstream-to=origin/master

# set foo as the tracking branch for origin/bar
git branch –track foo origin/bar

# update local tracking branches with changes from their respective remote ones
git fetch

#  update local tracking branches and remove local references to non-existent remote branches
git fetch -p

# delete remote tracking branch origin/branch
git branch -r -d origin/branch

# update local tracking branches and merge changes with local working directory
git pull

# given one or more existing commits, apply the change each one introduces, recording a new commit for each. This requires your working tree to be clean
git cherry-pick commitid

# push HEAD to the upstream url
git push

# push HEAD to the remote named origin
git push origin

# push HEAD to the branch master on the remote origin
git push origin master

# push and set origin master as upstream
git push -u origin master

# delete previous commits and push your current one
# WARNING: never use force in repositories from which other have pulled [1]
# https://stackoverflow.com/a/16702355

git push --force all origin/master

#
git push --force-with-lease

# turn the head of a branch into a commit in the currently checked out branch and merge it
git merge --squash mybranch
```

Going back by working with the History

```
# figures out the changes introduced by commitid and introduces a new commit undoing them.
git revert commitid

# does the same but doesn't automatically commit
git revert -n commitid

# updates the index and the HEAD to match the state of commit id.
# changes made after this commit are moved to “not yet staged for commit”
git reset commitid

# sets only the HEAD to commitid
git reset --soft commitid

# sets the HEAD, index and working directory to commitid
git reset --hard commitid

# sets the HEAD, index and working directory to origin/master
git reset --hard origin/master
```

Working with the Stash

```
# take all changes made to working tree and stash them in a new dangling commit, putting the working tree in a clean state
# DISCLAIMER: this does not include untracked files
git stash

# stash everything into a dangling commit, including untracked files
stash save --include-untracked

# apply the changes which were last stashed to the current working tree
git pop

# show the stash of commits
git stash list

# apply a particular commit in the stash
git stash apply

# apply the second-to-last commit in the stash
git stash apply stash@{2}

# drop the second-to-last commit in the stash
git stash drop stash@{2}

# stash only the changes made to the working directory but keep the index unmodified
git stash --keep-index

# clear the stash
git stash clear
```

Working with Submodules

```
# add a submodule to a repository and clone it
git submodule add https://domain.com/user/repository.git submodules/repository

# while in a repository which cointains submodules, they can be recursively updated by issuing the following command
git submodule init
git submodule update

# this an faster way of updating all submodules
git submodule update --init --recursive

# clone a repository which contains references to other repositories as submodules
git clone --recursive

# remove completely a submodule
submodule='mysubmodule';\
git submodule deinit $submodule;\
rm -rf .git/modules/$submodule;\
git config --remove-section $submodule;\
git rm --cached $submodule
```

Searching

```
#list the latest tagged revision which includes a given commit
git name-rev --name-only commitid

# find the branch containing a given commit
git branch --contains commitid

# show commits which have been cherry-picked and applied to master already
git cherry -v master

# look for a regular expression in the log of the repository
git show :/regex
```

Other Tips and Tricks

ls-files and ls-tree

```
# list the files contained in the current HEAD or in the head of the master branch respectively
git ls-tree --full-tree -r HEAD
git ls-tree -r master --name-only
git ls-tree -r HEAD --name-only

# list ignored files
git ls-files -i
```

Diffing

```
# diff two branches
git diff branch1..branch2

# perform a word-diff instead of a line-diff
git diff --word-diff

git diff --name-status master..branchname
git diff --stat --color master..branchname
git diff > changes.patch
git apply -v changes.patch
```

Cleaning

```
# perform a dry run and only list what untracked files or directories  would be removed without actually doing so
git clean -n

#remove untracked files from the working tree
git clean -f

# removes untracked files and directories
git clean -f -d

# same as above but also removes ignored files
git clean -f -x -d

# same as above but does so through the entire repo
git clean -fxd :/
```

git log one-liners

```
git whatchanged myfile
git log --after="MMM DD YYYY"
git log --pretty=oneline
git log --graph --oneline --decorate --all
git log --name-status

git log --pretty=oneline --max-count=2
git log --pretty=oneline --since='5 minutes ago'
git log --pretty=oneline --until='5 minutes ago'
git log --pretty=oneline --author=<name>
git log --pretty=oneline --all

git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short

git log --grep regexp1 --and --grep regexp2
git log --grep regexp1 --grep regexp2
git grep -e regexp1 --or -e regexp2
```

Useful BASH Aliases

Once you become familiar with the way git operates and are confident enough with the most common commands, I'd suggest giving aliases a try. They reduce the typing while using git quite a bit.

You can include the following in your `.bash_aliases` file.

```
alias g='git'
alias gs='git status '
alias ga='git add '
alias gb='git branch '
alias gc='git commit'

alias gf="git add .; git -c color.status=false status \
| sed -n -r -e '1,/Changes to be committed:/ d' \
            -e '1,1 d' \
             -e '/^Untracked files:/,$ d' \
             -e 's/^\s*//' \
             -e '/./p' \
| git commit -F -; git push"

alias gd='git diff'
alias go='git checkout '
alias gk='gitk --all&'
alias gx='gitx --all'
alias got='git '
alias get='git '
```

Set an SSH key for git access

```
ssh-keygen -t rsa -C "user@server.com"
cat id_rsa.pub

#remote of the repository must point to ssh url
git remote set-url origin ssh://git@server.com:/repo.git

#don't forget to upload your public key to the respective server
```

Now the following can be put inside `~/.ssh/config`.

```
host server.com
 HostName server.com
 IdentityFile ~/.ssh/id_rsa_server
 User git
```

List all dangling commits

```
git fsck --no-reflog | awk '/dangling commit/ {print $3}'
```

Leave the current commit as the only commit in the repository

```
git checkout --orphan new
git add -A
git commit -am "Initial commit"
git branch -D master
git branch -m master
```

Remove a file from the repository

```
git filter-branch -f --prune-empty --index-filter \
  'git rm --cached -r -q -- . ; git reset -q $GIT_COMMIT -- myfile' -- --all
```

Create a Repository on Gitlab using the API for every Directory in a List

```
#
for x in `ls|tr -d ' '`;\
do echo "creating $x ... ";\
curl -H "Content-Type:application/json" https://gitlab.com/api/v3/projects?private_token=REPLACE_WITH_VALID_TOKEN -d "{ \"name\": \"$x\" }";\
done
```

Set up a Git Repository using Git LFS

```
git init
git remote add origin git@domain.com:user/repository.git
git lfs track "*.jpg"
git lfs track "*.mpg"
git lfs track "*.mp4"
git lfs track "*.png"
git lfs track '*.bin'
git lfs track '*.iso'
git lfs track '*.zip'
git lfs track '*.rar'
git lfs track '*.7zip'
git lfs track '*.tar.gz'
git lfs track '*.gz'
git lfs track "*.avi"
git lfs track "*.pcap"
git lfs track "*.pcapng"
git lfs track "*.exe"
git lfs track "*.bmp"
git lfs track "*.bak"
git lfs track "*.bk"
git lfs track "*.obj"
git lfs track "*.odt"
git lfs track "*.pptx"
git lfs track "*.ppt"
git lfs track "*.doc"
git lfs track "*.docx"
git lfs track "*.xls"
git lfs track "*.xlsx"
git lfs track "*.dll"
git lfs track "*.o"
git lfs track "*.pdf"
git lfs track "*.msi"
git lfs track "*.jar"
git lfs track "*.ico"
git lfs track "*.gif"
git lfs track "*.tar"
git lfs track "*.bin"
git lfs track "*.data"
git lfs track "*.wmv"
git lfs track "*.dat"
git lfs track "*.db"
git lfs track "*.pickle"
git lfs track "*.csv"
git lfs track "*.list "
git lfs track "*.pyc"
git add .
git commit -m "Initial commit"
git push -u origin master
```

Githooks

```
# git hooks are scripts which can be executed after an action is performed, the options are: applypatch-msg ,commit-msg ,post-update ,pre-applypatch ,pre-commit ,prepare-commit-msg ,pre-push ,pre-rebase ,update

# git hooks for a given repository are stored under `.git/hooks`
[~/gitrepo]$ ls .git
branches  COMMIT_EDITMSG  config  description  FETCH_HEAD  gitweb  HEAD  hooks  index  info  logs  modules  objects  ORIG_HEAD  refs

[~/gitrepo]$ cd .git/hooks/

[~/gitrepo]$ ls -1

applypatch-msg.sample
commit-msg.sample
post-update.sample
pre-applypatch.sample
pre-commit.sample
prepare-commit-msg.sample
pre-push.sample
pre-rebase.sample
update.sample
```

Useful Githook Examples

Output something to /tmp/ before pushing

```
# create a simple pre-push githook which outputs to /tmp/githook.date.out
echo 'echo "pre-push" > /tmp/githook.`date +%s`.out' > .git/hooks/pre-push; chmod +x ./git/hooks/pre-push
```

prevent yourself from polluting the master branch by using commit

```
if [ $(git symbolic-ref HEAD 2>/dev/null) == "refs/heads/master" ]
then
  echo "Cannot commit to master branch"
  exit 1
fi
exit 0
```