Git Cheat Sheet

- Legend: `<>` required, `[]` optional

Create & Clone

- Clone an existing repository

```
git clone ~/git/some_local_repo.git &&
 git clone git@example.com:repo.git
```

default protocoll is ssh

- Create a new local repository

```
mkdir repo && cd repo && git init
```

- Import existing local repository to remote

```
git remote add origin ssh://git@exampl
e.com/repo.git && git push -u <remote>
 <branch>
```

`-u` *≙ push for first time and set upstream*

- Create remote repository for existing local data

```
mkdir repo.git && cd repo.git && git i
nit --bare
```

The remote repository has to be »bare« (does not contain a working filetree, but a special .git subdirectory only) in order to accept a push

Show local changes

- Show status - current branch name and changed files in working directory

```
git status
git st
```

`git st` *requires an alias set for »st«*

- Show all changes not yet staged (in tracked files only)

```
git diff
```

- Difference between staged files and HEAD

```
git diff --cached
```

- Difference between all files and HEAD since last commit

```
git diff HEAD
```

- Show only names of changed files, e.g. to show all file differences between two branches

```
git diff master --name-only
git diff master --name-status
```

- Difference between branches, two commits, etc

```
git diff <foo> <bar>
```

»+« line does exist in »bar« but not in »foo«, »-« reverse

Show history

- Show all commits, starting with newest

```
git log
```

- Show history of changes with files changed

```
git whatchanged
```

- Show changes over time for a specific file

```
git log -p <file>
```

- Show who changed what and when in a file

```
git blame <file>
```

left side shows the last commit ID for the content on the right side

- Show a commit identified by its ID

```
git show <ID>
```

- Show all commits with a certain word in commit message

```
git log --grep=<searchword>
```

Commit

- Add all (even untracked) changed files to next commit (≙ stage)

```
git add -A
```

- Stage a tracked and modified file

```
git add <file>
```

- Commit all local changes in tracked files

```
git commit -a
```

- Add hand-picked changes in a file to the next commit (partial commit)

```
git add -p <file>
```

`y` *Add this part to the next commit*
`n` *Skip this part*
`d` *Don't add this and all remaining parts*
`s` *Split the parts info smaller ones (only works if there's unchanged lines between the changes in the displayed part)*
`e` *Manually edit the parts*

- Commit previously staged changes

```
git commit
git commit -m "<message>"
```

- Change the last commit

```
git commit --amend -a
```

Branches

- List local branches

```
git branch
```

`*` *marks the current branch*

- List remote-tracking branches

```
git branch -r
```

use `-a` *to show local and remote-tracking branches at once*

- Switch to a different branch

```
git checkout <branch>
git checkout -t <remote>/<remote-branc
h>
```

`-t` *checkout a new branch based on remote branch and track it*

- Create a new branch based on your current HEAD

```
git branch <new-branch>
```

use `git checkout -b <branch>` *to create a branch and switch right into it*

- Create a new branch based on a remote branch and track it

```
git branch --track <new-branch> <remot
e>/<remote-branch>
```

use `--no-track` *to create a new branch based on a remote branch but don't track it*

- Set remote-tracking branch for current local branch

```
git branch --track <local-branch> <rem
ote>/<remote-branch>
```

- Show merged branches

```
git branch -a --merged
```

`--no-merged` *will show branches not merged yet*

- Delete a local branch

```
git branch -d <branch>
```

`-d` *will only delete the branch if it is merged with its upstream branch (if set),* `-D` *will force the deletion*

- Delete a remote branch

```
git push <remote> :<remote-branch>
```

Tags

- Show all tags

```
git tag -n
```

`-l` *will show tag names only,* `-n<num>` *will add a number of lines from the annotation (default is one)*

- Mark the current commit with a tag

```
git tag -m "<annotation>" <tag-name>
```

Update

- Download all changes from , but don't merge them into HEAD

```
git fetch <remote>
```

a manual merge is required now

- Download changes and directly merge/ integrate into HEAD

```
git pull [<remote> <branch>]
```

if the remote branch is tracked then `git pull` *is sufficient*

- List all currently configured remote repositories

```
git remote -v
```

- Show information about a remote, e.g. which branches exist in this remote

```
git remote show <remote>
```

- Deletes all stale remote-tracking branches in a remote

```
git remote prune <remote>
```

- Add a new remote repository

```
git remote add <remote> <url>
```

Publish

- Publish local changes in a branch to a remote repository

```
git push [<remote> <branch>]
git push --set-upstream <remote> <bran
ch>
```

push branch to remote and track it, `-u` *is the short option*

- Publish all local branches to a remote

```
git push --all <remote>
```

- Publish a tag to a remote

```
git push <remote> <tag>
```

- Publish your all tags

```
git push --tags <remote>
```

Merge

- Merge a branch into your current HEAD

```
git merge <branch>
```

- Manually solve conflicts and mark file as resolved

```
git add <resolved-file> && git commit
-m 'Manual Merge'
```

- Use a merge tool to solve conflicts

```
git mergetool -t meld
git mergetool
```

if empty this will use the configuration variable »merge.tool«

- Use a recursive merge strategy

```
git merge -s recursive -X <ours|theirs
|patience>
```

»recursive« is the default merge strategy when pulling or merging one branch, so it may be dropped
»ours« will prefer current branch head over conflicting changes from the other tree but merge non-conflicts normally
»patience« will cause GIT to spend some extra time while merging to avoid merge mistakes

- Force merge to use current branch head and ignore all changes from other branches

```
git merge -s ours
```

- Cancel merge

```
git merge --abort
```

- Undo a local merge

```
git reset --hard <ID>
git reset --hard <remote>/<branch>
```

resetting to remote branch will only work if the remote branch isn't ahead of your local one
to undo a merge that was already pushed you'll have to do a revert

Rebase

- Rebase your current HEAD onto

```
git rebase <branch>
```

!! Don't rebase published commits

- Abort a rebase

```
git rebase --abort
```

- Continue a rebase after resolving conflicts

```
git rebase --continue
```

Stash

- Stash all changes away

```
git stash save [<annotation>]
```

- Show all available stashes

```
git stash list
```

»stash@{0}« is the keyname of the stash, where 0 is the latest

- Apply state from the stash list to the current working tree

```
git stash apply [<stash-name>]
```

default is »stash@{0}«, use `git stash pop` *to apply changes and remove the state from stash list*

- Remove a state from the stash list

```
git stash drop [<stash-name>]
```

- Remove all the stashed states

```
git stash clear
```

Revert

- Remove files from stage (unadd)

```
git reset HEAD
git reset HEAD -- <single-file>
```

- Discard local changes in a specific file

```
git checkout <file>
```

- Discard all local changes in your working directory

```
git reset --hard HEAD
```

- Revert a commit (e.g. to revert a commit already pushed to remote)

```
git revert HEAD
git revert <commit>
```

produces a commit (alters the history) with contrary changes to last commit

- Remove a file from index but keep it on file system

```
git rm --cached
```

Configuration

- Get configuration option

```
git config <section>.<key>
```

- Set configuration option

```
git config --local <section>.<key> <va
lue>
```

»local« will write to ».git/config« in current repository, »global« to »~/.gitconfig» and »system« to your systems »/etc/gitconfig«

- Set username and e-mail

```
git config --local user.name "<usernam
e>" && git config --local user.email <
e-mail>
```

- Ignore mode changes (chmod)

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

- Set alias »st« for »status«

```
git config --global alias.st status
```

Commit Message Format

```
[BUGFIX] Short summary

Optional explanatory text. Separated by
new line. Wrapped to 74 chars. Written i
n imperative present tense ("Fix bug", n
ot "Fixed bug").

Help others to understand what you did (
Motivation for the change? Difference to
 previous version?), but keep it simple.

Mandatory title prefix: [BUGFIX], [FEATU
RE] (also small additions) or [TASK] (no
ne of the above, e.g. code cleanup). Add
itionall flags: [!!] (breaking change),
[DB] (alter database definition), [CONF]
 (configuration change), [SECURITY] (fix
 a security issue).

Bug tracker refs added at bottom (see ht
tp://is.gd/commit_refs).

Resolve #42
Ref #4 #8 #15 #16
```

shortened, detailed example at http://is.gd/commitformat

Best practices

- Commit related changes
  - Each commit should adress one logical unit. Two different bugs should result into two commits.
- Commit early & often
  - Keep your commits small and comprehensible, split large features into logical chunks.
- Test code before committing
  - Make shure the code works, don't guess. Revert the commit if necessary.
- Don't commit half-done work
  - Commit only complete, logical changes, not half-done chunks. »Stash« changes if applicable.
- Don't commit hot files
  - Don't commit configuration files (commit a config template instead), personal data, temporary files (GIT is no backup system) or things that can be regenerated form other commited things.
- Write good commit messages
  - Help others to understand what you did (Motivation for the change? Whats the difference to the previous version?)
  - Useless commit messages may be forwarded to whatthecommit.com
- Write in imperative present tense («change», not «changed» or «changes»)
  - A commit is a set of instructions for how to go from a previous state to a new state, so you should describe was the commit does and not what it did to your repository.
- Don't panic
  - GIT lets you undo, fix or remove a bunch of actions
- Don't change published history
  - GIT allows you to rewrite public history, but it is problematic for everyone and thus it is just not best practice to do so.
- Use branches
  - Branching is cheap. Use separate branches for bugfixes, features & ideas. Make branching a part of ypur local workflow.
- Merge regularly
  - Don't merge a huge feature into the master, instead merge the master regularly with your branch.
- Use conventions
  - As with every development process: use conventions. For naming of branches and tags, how to write commit messages, when to commit into what branch, etc.

Sources

- Git Cheat Sheet by Ying Guo
- Git Cheat Sheet by Git Tower
- http://gitready.com/
- http://git-scm.com/documentation
- http://wiki.typo3.org/CommitMessage_Format_(Git)

About

- Supervisor: Dan Untenzu @pixelbrackets
-
- Disclaimer: I Have No Idea What I'm Doing
- Flattr: https://flattr.com/profile/pixelbrackets