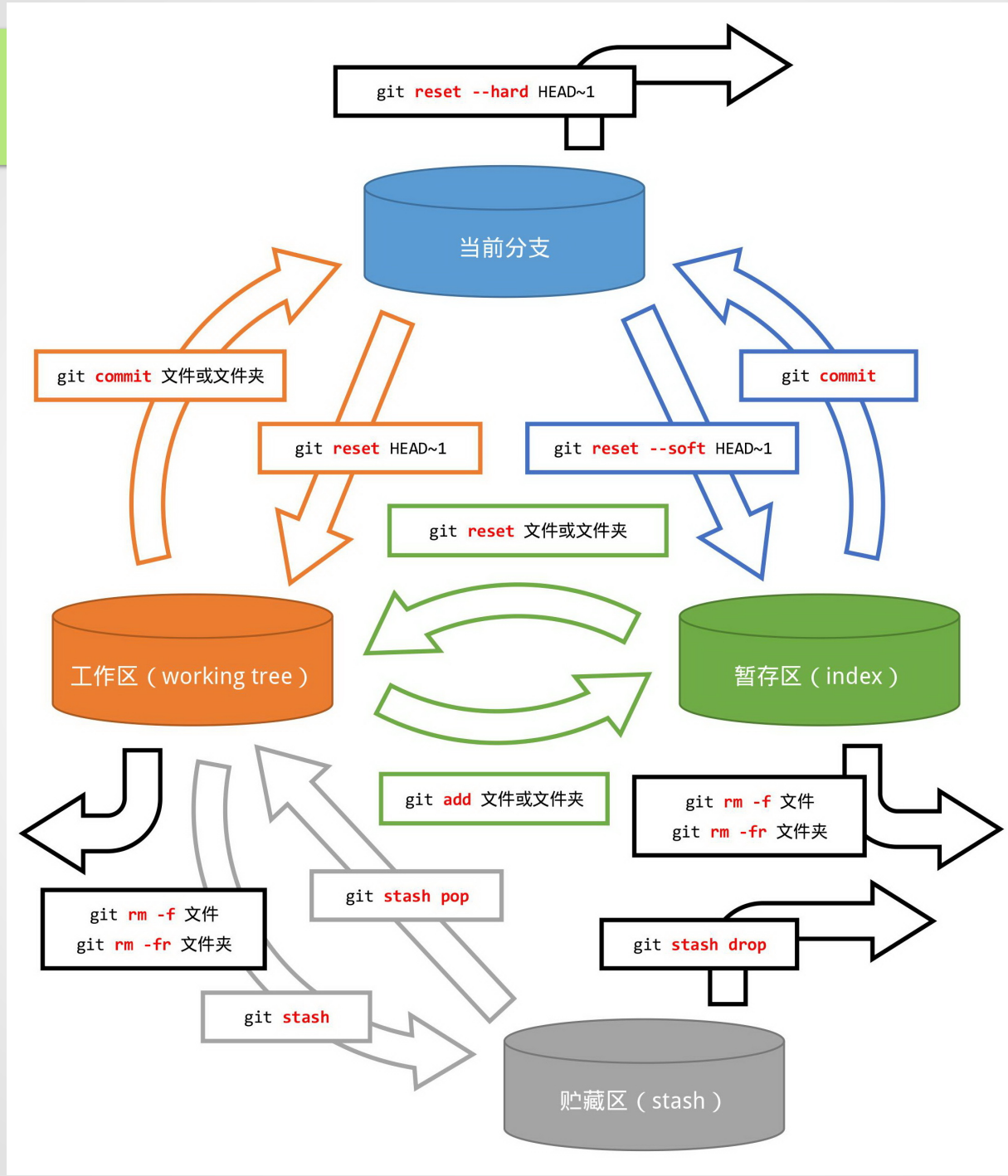
Git 命令速查

- 文件操作
- 分支操作
- 仓库操作

文件操作

Git 仓库目录结构

- 工作区（ *working tree* ）
  - 本地已修改，但是未进行标记
    - 未被版本控制追踪的文件不在此列
    - 包含使用非 Git 的工具删除的文件
- 暂存区（ *index* 或 *staging area* ）
  - 本地已修改，并且标记为待提交
    - 包含使用 Git 删除的文件
- HEAD


git reset --hard HEAD~1
当前分支
git commit 文件或文件夹
git reset HEAD~1
git reset --soft HEAD~1
git commit
git reset 文件或文件夹
工作区（working tree）
暂存区（index）
git add 文件或文件夹
git rm -f 文件
git rm -fr 文件夹
git rm -f 文件
git rm -fr 文件夹
git stash pop
git stash drop
git stash
贮藏区（stash）

工作区 ⇒ 暂存区

- 从工作区增加文件到暂存区（标记为待提交）
  - `git add 路径...`
    - 如果是被 .gitignore 忽略的文件，使用 -f 强制添加
  - `git add -A 路径...`
    - 增加指定目录下的所有未被忽略的文件

暂存区 ⇒ 工作区

- 从暂存区取消待提交的文件到工作区
  - `git rm --cached 路径 ...`
    - 取消待提交的文件夹需要使用 `-r` 参数
  - `git reset 路径 ...`
- 从暂存区恢复文件到工作区
  - `git checkout 路径 ...`

暂存区 ⇒ HEAD

- 使用暂存区的文件创建 commit
  - `git commit`
    - 没有路径参数。

HEAD ⇒ 暂存区

- 撤销最后的 commit，其中的修改进入暂存区
  - `git reset --soft HEAD~`
- 撤销最后的 3 个 commit
  - `git reset --soft HEAD~3`
- 从 HEAD 恢复文件到暂存区并丢弃工作区中的修改
  - `git checkout HEAD 路径 ...`

工作区 ⇒ HEAD

- 使用工作区的文件创建 commit（跳过暂存区提交）
  - `git commit 路径...`
    - 必须指定至少一个文件或文件夹。
- 把工作区中已修改的文件和暂存区中的内容一起提交
  - `git commit -a`

HEAD ⇒ 工作区

- 撤销最后的 commit，其中的修改进入工作区
  - git reset HEAD~

删除文件

- 删除工作区和暂存区中的文件
  - `git rm 路径 ...`
    - 如果文件有未提交的修改，使用 `-f` 参数强制删除
    - 删除文件夹需要使用 `-r` 参数
    - 删除之后在暂存区中标记为待删除，仍然需要提交

分支操作

创建分支

- 以 HEAD 创建本地分支，并切换到新分支
  - git checkout -b 新分支
    - 如果新分支已存在而要强制覆盖，使用 -B 参数
- 以 HEAD 创建本地分支，但是不切换到新分支
  - git branch 新分支

删除分支

- 删除本地分支
  - `git branch -d 待删除分支`
    - 如果分支未合并而要强制删除，使用 `-D` 参数

重命名分支

- 重命名本地分支
  - `git branch -m 旧分支 新分支`
    - 如果新分支已存在而要强制覆盖，使用 `-M` 参数

推送分支

- 将当前分支推送到其上游分支
  - `git push`
- 将指定本地分支推送到其上游分支
  - `git push 远程仓库 本地分支 ...`
- 将本地 `ref` 推送到远程仓库中的分支
  - `git push 远程仓库 ref:分支 ...`

合并修改： merge

- master　　A — B — C — D — E
- dev-1　　A — B — M — N
- dev-2　　A — B — C — X — Y

- 当前分支是 dev-2

git merge dev-1

- master　A — B — C — D — E
- dev-1　A — B — M — N
  dev-2　A — B — C — X — Y ┬ R
  　　└— M — N ——┘
- 当前分支是 dev-2
- git merge 不会重写历史

合并修改: rebase

- master　　A — B — C — D — E
- dev-1　　A — B — M — N
- dev-2　　A — B — C — X — Y

- 当前分支是 dev-2

git rebase dev-1

- master　　A — B — C — D — E
- dev-1　　A — B — M — N
- dev-2　　A — B — M — N — C'— X'— Y'

- 当前分支是 dev-2
- git rebase 会重写当前分支上最后的一部分历史

合并修改： rebase onto

- master　　A — B — C — D — E
- dev-1　　A — B — M — N
- dev-2　　A — B — C — X — Y

- 当前分支是 dev-2

git rebase dev-1 --onto=master

- master　　A — B — C — D — E
- dev-1　　A — B — M — N
- dev-2　　A — B — C — D — E — X'— Y'

- 当前分支是 dev-2

合并修改： cherry-pick

- master　　A — B — C — D — E
- dev-1　　A — B — M — N
- dev-2　　A — B — C — X — Y

- 当前分支是 dev-2

git cherry-pick N D M

- master　A — B — C — D — E
- dev-1　A — B — M — N
- dev-2　A — B — C — X — Y — N'— D'— M'

- 当前分支是 dev-2
- git cherry-pick 类似于 svn merge

仓库操作

更新本地仓库

- 更新当前分支的上游仓库数据到本地
  - `git fetch`
    - 使用 `-p` 参数清理上游已被删除的数据
- 更新指定远程仓库上数据到本地
  - `git fetch` 远程仓库
- 更新所有远程仓库上数据到本地
  - `git fetch --all`

更新当前分支和对应的上游仓库

- 更新当前分支的上游仓库，然后执行 git merge
  - `git pull`
- 更新当前分支的上游仓库，然后执行 git rebase
  - `git pull -r`