Vim项目目录下

Apache

```
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=1 ../..
```

> https://gitee.com/CloudGuan/nvim-for-server?_from=gitee_search
>
> VIM操作

指令

· ctrl+p

> 作用：Leaderf文件搜索，相当于小番茄的alt+shift+o
>
> 位置：nvim/setting/leaderf.vim
>
> 技巧：
>
> 如果新加的文件搜不到，按**F5**
>
> ctrl+j/k 选择文件后ctrl+T可以在新页签打开.可以自己映射其他快捷键在多页签切换，比如我的:

HTML

```
" 前一个 后一个 tab
noremap <C-h> :tabp <cr>
noremap <C-l> :tabn <cr>
```

· <leader>+p (,p)

> 作用：列出当前文件的函数，相当于vs的alt+m
>
> 位置：nvim/setting/leaderf.vim

· gd,gr

> 作用：*gd* 跳转至定义 *gr* 列出引用
>
> 位置：nvim/setting/cocconf.vim

· Ctrl + o ， Ctrl + I

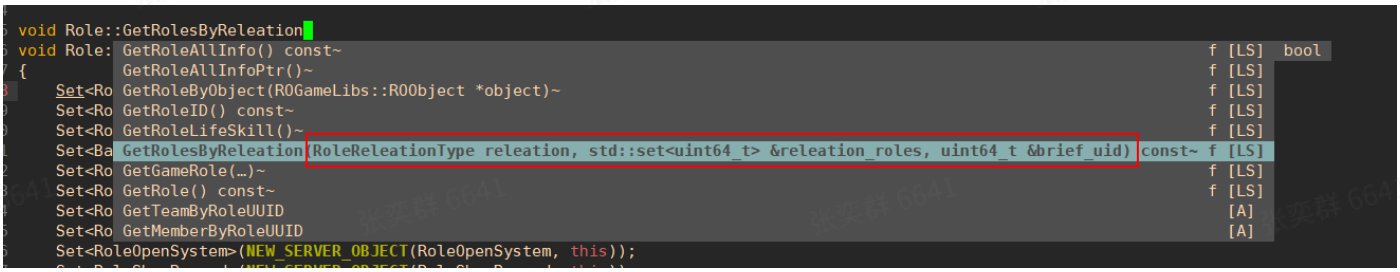
作用：往后/前跳转，

· F2

> 作用：打开目录
>
> 位置：nvim/setting/defx.vim
>
> 目录和代码切换的快捷键命令：jjkk
>
> ctrl+w+h:回到目录界面
>
> ctrl+w+l:回到右边的代码界面

· 函数参数补全

> 输入函数前几个字符时，下面会列出提示，ctrl+N/P可以上下移动选择，选中之后按ctrl+y可以确认选择并且补全函数参数，还会包含相应头文件.可以用于：在头文件定义了函数，要到cpp实现函数的时候使用。



插件

需要加在~/.config/nvim/init.vim中

如果在编辑模式添加，关闭文件后又会失效

· DoxygenToolkit

> 作用：自动生成注释
>
> 添加：Plug 'vim-scripts/DoxygenToolkit.vim'
>
> 使用：光标移至函数上一行，命令模式下输入`:Dox`
>
> 附件：

Doxyten

· Gtags

> 作用：代码引用搜索
>
> 添加：Plug 'gtags.vim''
>
> 使用：参考https://www.gnu.org/software/global/教程

> 附件：https://www.gnu.org/software/global/

- grep.vim

> 作用：关键字快速搜索
>
> 添加：Plug 'grep.vim''
>
> 使用：命令模式:Rgrep
>
> 附件：https://www.vim.org/scripts/script.php?script_id=311

- asyncrun.vim

> 作用：异步代码执行这个
>
> 添加：Plug 'skywind3000/asyncrun.vim''
>
> 使用：命令模式:asyncrun xxx
>
> 附件：https://www.vim.org/scripts/script.php?script_id=311

常用插件地址

https://github.com/vim-scripts/DoxygenToolkit.vim

https://github.com/universal-ctags/ctags

https://ftp.gnu.org/pub/gnu/global/global-6.6.tar.gz

常用脚本

当clone的时候有指定分支，需要新拉一个已存在的分支时：

两个参数,已在在的分支名和拉的depth.在工程根目录执行

Bash

```bash
#!/bin/bash

if (( $# < 2 )); then
    echo "需要两个参数：分支名和日志深度"
    exit 1
fi

branch=${1}
depth=${2}


cat .git/config | grep "refs/heads/${branch}:refs/remotes/origin/${branch}" >/dev/null
if (( $? > 0 )); then #说明没添加
    git remote set-branches --add origin ${branch}
fi

git branch -r | grep origin/${branch} > /dev/null
if (( $? > 0 )); then # 说明没有拉下来
    git fetch origin ${branch}:${branch} --depth ${depth}
    if (( $? > 0 )); then
        echo "git fetch 失败，请手动重试"
        exit 1
    fi
fi

git checkout ${branch}
if (( $? > 0 )); then
    echo "git checkout ${branch} 失败,请手动重试"
    exit 1
fi
git push -u origin ${branch}

echo "Done"
```

修改东西太多，想要一个个提交确认：

空格表示要提交，回车表示跳过

Bash

```bash
iles=$(git status | grep -v "# On branch" | grep -v "\"git")

# 最终要add的文件
final_add=""
```

```
final_rem=""
function try_read()
{
    if [[ $# < 2 ]]; then
        return
    fi
    while read -n1 rlt; do
        if [[ -z ${rlt}
            || ${rlt} == ' ' ]]; then
            break;
        fi
    done
    if [[ ${rlt} == ' ' ]]; then
        if (( ${2} == 1 )); then
            final_rem="${final_rem} ${1}"
        else
            final_add="${final_add} ${1}"
        fi
    fi
}

IFS=$'\n'
not_stage_str="# Changes not staged for commit:"
untrack_str="# Untracked files:"
# 过虑掉已add过的内容
for fd in ${files[@]}; do
    if [[ ${fd} == ${not_stage_str}
        || ${fd} == ${untrack_str} ]]; then
        break
    fi
done

# 判断是否无视的文件
ignore_files_reg=(^.cache/
    ^G
    ^rogamelib
    ^lib/
    ^vimjson/
    ^buildtool
    ^tscancode
)

ignore_files=("$"
    "gitadd.sh"
    "compile_commands.json"
    "add_exist_branch.sh"
)
```

```
function CheckIgnore()
{
    if [[ $# < 1 ]]; then
        return 1
    fi
    for ig in ${ignore_files[@]}; do
        if [[ ${1} == ${ig} ]]; then
            return 1
        fi
    done

    for ig in ${ignore_files_reg[@]}; do
        if [[ ${1} =~ ${ig} ]]; then
            return 1
        fi
    done
    return 0
}

# 处理修改中的内容
modify=0
modify_reg="^#.*modified:*"
delete_reg="^#.*deleted:*"
for fd in ${files[@]}; do
    if [[ ${fd} == ${untrack_str} ]]; then
        break;
    fi
    if [[ ${fd} == ${not_stage_str} ]]; then
        modify=1
        continue
    fi
    if (( modify == 1 )); then
        if [[ ${fd} == "#" ]]; then
            continue
        fi
        rfd=$(echo ${fd} | awk '{print $3}')
        CheckIgnore ${rfd}
        if (( $? != 0 )); then
            continue
        fi
        if [[ ${fd} =~ ${modify_reg} ]]; then
            echo "修改的文件: ${rfd}"
            try_read ${rfd} 0
        elif [[ ${fd} =~ ${delete_reg} ]]; then
            echo "删除的文件: ${rfd}"
            try_read ${rfd} 1
        else
            echo "未知类型文件:${fd}"
```

```
        fi
    fi
done

# 未追踪的文件(夹)
untrack=0
for fd in ${files[@]}; do
    if [[ ${fd} == ${untrack_str} ]]; then
        untrack=1
        continue
    fi
    if (( ${untrack} == 1 )); then
        if [[ ${fd} == "#" ]]; then
            continue
        fi
        rfd=$(echo ${fd} | awk '{print $2}')
        CheckIgnore ${rfd}
        if (( $? != 0 )); then
            continue
        fi
        echo "未添加文件(夹): ${rfd}"
        try_read ${rfd} 0
    fi
done

IFS=$' '
if (( ${#final_add} > 0 )); then
    git add ${final_add[@]}
else
    echo "没有要提交的"
fi
if (( ${#final_rem} > 0 )); then
    git rm ${final_rem[@]}
else
    echo "没有要删除的"
fi
echo "------------------------------------------------------------------------"

git status
```